The Robust Design of Complex Systems

Ronald Anthony Bates

Submitted as fulfillment for the degree of Ph.D.
City University
Engineering Design Centre

April 1995

Contents

List of Tables

List of Figures

Acknowledgement

I would like to thank my Ph.D advisor Henry Wynn and Bob Buck my unofficial mentor for all their help and encouragement during the course of studies embodied by this thesis. I must also acknowledge the support of the Engineering Design Centre at City University, particularly Henry Wynn and Alan Jebb, for providing a stimulating working environment and the financial support given by Mentor Graphics (UK) Ltd. and the Science and Engineering Research Council.
On a personal level I would like to thank my family, especially my parents, for their constant support and Natasha Robertson for bearing with me during the more stressful moments of writing-up.
I would also like to thank a number of people for their friendship and good advice: Henry Wynn for being a great boss, Bob Buck for letting me win so many games of pool, Tom Parsons for keeping things in perspective, Rick of Rye Wholefoods for many superb lunches and Eva Riccomagno for trying to teach me some maths.

Abstract

Robust Engineering Design has evolved as an important methodology for the integration of quality with the process of design. The methodology encompasses the disciplines of experimental design, model building and optimization. First an experiment is conducted on a system (or a simulation of the system), second a model is built to emulate the system and finally the emulation model is used to optimize the system design. Applying these methods to large problems can be difficult and time-consuming because of the complexity of most design problems. It is the goal of this thesis to introduce methods which reduce problem complexity and so make the application of Robust Engineering Design (RED) methodology easier for large design problems.

By drawing from methods used in systems theory and circuit optimization several techniques are presented with the aim of reducing the complexity of performing experiments for Robust Engineering Design. A common framework for experimentation is created by combining a commercial circuit simulator with established methods for experimental design and model building. This provides the basis for experimentation in subsequent chapters. A method of design optimization with respect to quality is presented to complete the model-based Robust Engineering Design cycle.

Three approaches to reducing problem complexity are adopted. First a method of system decomposition is applied directly to an electronic circuit to reduce the size of experiment required for RED. Second a method of modelling system response functions is described which integrates the action of the circuit simulator with the model building process. Third information about system topology is used in the design of experiments to enhance the model-building process.

Conclusions are drawn about the effectiveness of the approaches described with respect to the impact on problem complexity.

Preface

The format of this thesis is as follows :

Introduction. This is Chapter 1. The Introduction provides a grounding for the work contained in the thesis and includes a critique of circuit optimization and Robust Engineering Design.

Tools and techniques. This is Chapter 2. The section reviews the tools and techniques used throughout the thesis, including a review of circuit simulation, specific methods for Robust Engineering Design, system decomposition and algorithms for the decomposition of graphs.

Technical chapters. Comprising Chapters 3 to 7. The technical chapters are split into two parts. The first part deals with the application of Robust Engineering Design within a commercial environment including the development of an integrated system for Robust Circuit Design and a method of design optimization. The second part contains descriptions of techniques developed in an attempt to reduce the complexity of performing Robust Engineering Design with electronic circuit simulators. This covers techniques for modelling circuit response functions and two decomposition methods.

Conclusions. This is Chapter 8. The conclusion sums up the work described and outlines future possible work.

Appendices. The appendices contain the computer code developed and used in the thesis.

Where appropriate a section at the end of each chapter contains relevant bibliographic material to preserve the flow of the main text.

Chapter 1

Introduction

1.1 General Introduction

This thesis discusses techniques for analysing systems in order to make them robust against variation in manufacture and use. This is commonly referred to as Robust Engineering Design or RED. Particular attention is drawn to the problems of applying RED methodologies to electronic circuit design and to developing new ways of tackling large or complex RED problems.

1.1.1 Quality and Robust Engineering Design

Awareness of the need for quality in an increasingly competitive industrial environment has led to the development and use of new techniques for product design. Methodologies such as concurrent engineering highlight the need to consider all aspects of design, manufacture, production and use at the design stage of developing a product or process. Implementing such methods is a formidable task especially for large and complex products.

Robust Engineering Design (RED) encompasses part of this drive for high quality products. RED can be considered as a philosophy for designing products that are insensitive to variations in manufacture and use. Intrinsically linked is the definition of quality in terms of "fitness for purpose". The central idea is that by understanding and quantifying the environment in which a product is manufactured and used it can be designed to a certain quality (defined later in this section) . The objectives are to maximise quality and minimise cost.

Methods employed for RED draw heavily from the statistical community where definitions of variance, noise, error etc. are used to define variability in terms of a product or process and provide the basis for the formulation of a solution. The application of basic statistical methods to engineering problems is generally credited to Genichi Taguchi. Taguchi's main contribution was not in the invention of any particular statistical technique but rather in popularising a few techniques in the engineering community for use in solving engineering design problems. By using basic experimental design techniques and Analysis of Variance (ANOVA) tables a basic form of Robust Design can be easily applied by engineers to design problems. Taguchi is often criticised for over-simplifying RED and the techniques adopted by him are generally not considered good practice. However the definition of quality as "the loss to society once a product is shipped" remains a significant contribution along with measuring loss continuously as deviation from some target value.

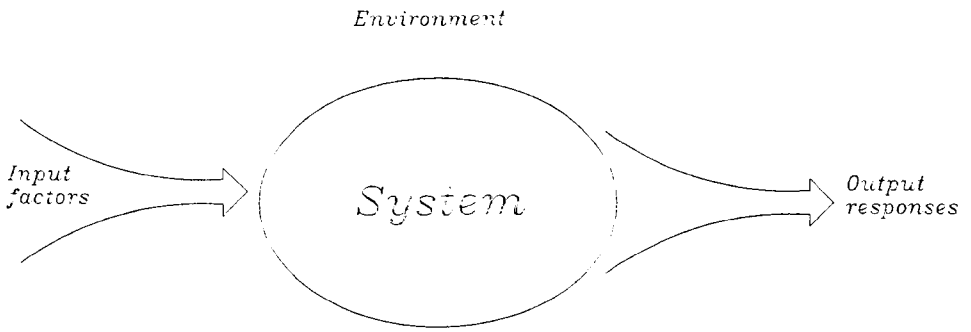


Figure 1.1: Generalised system

Definition of a system

For the purposes of this thesis a system is defined in a general sense as a function with inputs (factors), representing function parameters and signal inputs, and outputs (responses). This is summarised in Figure 1.1. It should be noted that this definition is in contrast to the state-space notation used in Engineering to mathematically model systems. Throughout the thesis we deal with empirical models and treat model parameters and signals equally as factors which affect system response.

Definition of quality

The goal of RED is to minimise the variability of design performance which we divide into two sources, variability in the parameters of the design, *internal noise* and variability due to outside influences, *external noise*. We define a system as a function f with

$$Y = f(X, Z) \tag{1.1}$$

Y being a vector of system responses or outputs, X being the vector of system parameters or factors under the control of the designer, called *control factors*, and Z is the vector of *noise factors* which are not under the control of the designer e.g temperature and humidity. We mimic product variability by defining the control and noise factors as independent random variables with probability density functions $g_1(X)$ and $g_2(Z)$. Letting $L(Y)$ be a *loss function* dependent on the response we define the *risk* as expected response

$$R = \int\int L(Y(X,Z))g_1(X)g_2(Z)dxdz = E(L(Y)) \tag{1.2}$$

A useful loss function is quadratic. Letting $\mathcal{T}$ be a target response vector the quadratic loss function is

$$L = (Y - \mathcal{T})^2. \tag{1.3}$$

If the control factors do not have any noise associated with them, for example if we want to determine the control factors for a particular design, an approximation of the risk becomes the mean squared error risk,

$$R = E(Y - \mathcal{T})^2 = \mathrm{Var}_{noise}(Y) + (E_{noise}(Y) - \mathcal{T})^2 \tag{1.4}$$

where Var_{noise} and E_{noise} mean with respect to the variation in Z. Thus the quality of a design in this thesis is defined in terms of the expected (i.e mean) performance and the variability about the mean.

$$\mathcal{Q} = f(E_{noise}(Y), \mathrm{Var}_{noise}(Y)) \tag{1.5}$$

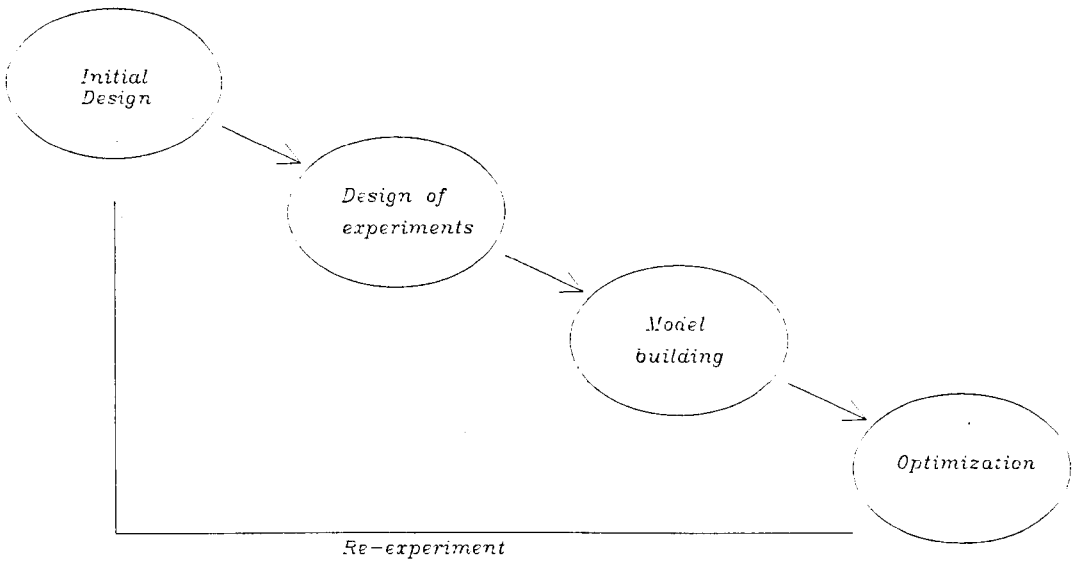


Figure 1.2: Schematic of RED process

In Chapter 4 this idea is extended to optimize systems using the criteria of attaining target while minimising variability, the approximate solution being

$$\min\left(\mathrm{Var}_{noise}(Y)\right) \tag{1.6}$$

subject to

$$E_{noise}(Y) = \mathcal{T} \tag{1.7}$$

This is referred to as the unbiased solution.

1.1.2 RED scheme

The fundamental scheme of RED applied in this thesis is summarised in Figure 1.2. The RED process is iterative, stopping when the optimization process produces a solution of the required accuracy. Increased accuracy is obtained by reducing the number of input

factors in the experiment and building the model over a smaller space i.e reducing the range of the input variables. The basic steps are:

i. *Design of Experiments.* Experimental Design is described in Section 1.3.2. The idea is to choose an experimental design plan which defines the parameter values for the system of interest in a series of trials where the system response is measured.

ii. *Model building.* Mathematical models are fitted to data collected from the experimental designs.

iii. *Optimization.* The models are used to predict system response as part of a numerical optimization procedure.

iv. *Redesign.* Once optimized the system is verified. If unsatisfactory the system is redesigned and the RED process repeated.

1.1.3 System complexity

System theory is a vast and variously defined subject. One definition is that every real physical process is a system. Examples include linear systems in control, non-linear dynamic systems, biological and chemical systems and 'soft system' methodologies in management science [24].

Intimately related to the idea of a system is that of complexity, again defined in many different ways.

Definition

The definition we use in this thesis is that a system is complex if it contains many interacting elements or subsystems which can exist in different states.

Very importantly, complexity can also be taken as a measure not so much of the system itself as of our ability to learn about it. This provides a conceptual link with Chapter 7 where knowledge about the system is used to shape the way we observe it i.e the way the experimental design plan is structured.

The main difficulty in applying RED to complex problems is that the size of experiment increases rapidly with problem size. For a complex system with lots of input factors the experimental design plan can be prohibitively large in its attempt to effectively fill the input space. However there exists a wide body of material on dealing with large systems which is discussed in Section 2.7.

Large systems require new methods of experimental designs suitable for the highly adaptive models which are employed to cope with complex non-linear responses and high dimensionality of input spaces. The area of computer experiments has started to provide such designs especially Latin Hypercube and Lattice designs. System decomposition, prevalent in several branches of engineering, can be employed to decrease complexity.

The high dimensionality of input and output spaces of many systems presents special problems in experimental design. Traditional methods, notably factorial design (Section 1.3), have gone some way towards meeting the challenge. For example complex industrial quality control has stimulated renewed interest in highly fractional designs at least as an initial screening for significant factors.

When a physical process is modelled by a large simulator, such as for an electronic circuit or a finite element analysis of stress on a mechanical product, experiments can be conducted directly on the computer code. This leads to the subject of the design and

analysis of computer experiments, DACE, which is a rapidly growing area of experimental design [35, 31, 34, 15, 50]. The complexity and nonlinearity of the code has meant that factorial and response surface method have given way to two new methodologies (i) fitting highly adaptive models (ii) the use of 'space-filling' experimental designs. This thesis draws heavily on this area of research in trying to make suggestions about how to experiment on large systems.

We relate the notion of complexity to systems theory. In a general sense we think of something as being complex if we do not understand it or we cannot deal with it at that time. In systems theory complexity can be used to measure the size of a system or problem, where its complexity depends not only on size but also on our ability to deal with it. For example a problem may be complicated in terms of the number of variables and interactions but be handled easily by a specialised software routine; in this case the problem would not have a high degree of complexity. The development of ϵ-complexity, defined as the time taken for the fastest algorithm to solve a given problem to within a certain error bound ϵ, shifts the emphasis from the problem to the algorithm used to solve it. See [49] for an example.

1.1.4 System decomposition

A common theme for reducing complexity is the idea of system decomposition, namely that the system can be considered as a collection of interacting subsystems. Decomposition methods are used in the design, analysis, control and optimization of systems to allow complex problems to be handled efficiently. An important distinction in this thesis is between decomposition in a physical sense referred to as *partitioning* and

decomposition for mathematical analysis referred to as *tearing*.

Definition

Partitioning involves breaking up of a graph or network of the system, physically decomposing a network. This implies that no system equations have yet been formed and that separate equations will result from the decomposition process.

Definition

Tearing requires that the system equations be stated in full before decomposition, such as the formulation of a sparse matrix in block form, can take place i.e for purposes such as sparse matrix methods. Through working with electronic circuits we have been drawn to using methods from this area.

Section 2.7 reviews different decomposition techniques.

1.1.5 RED for complex systems

In this thesis methods are presented to make RED for large systems easier to perform. The approach is first to integrate existing RED methods in a single package and second to adapt ideas from systems theory to reduce the complexity of applying RED to large systems. The methods are applied exclusively to electronic circuits but are readily generalised to other engineering systems.

The principal lesson which emerges is the following:

in conducting an experiment on a subsystem of a complex system it is essential to

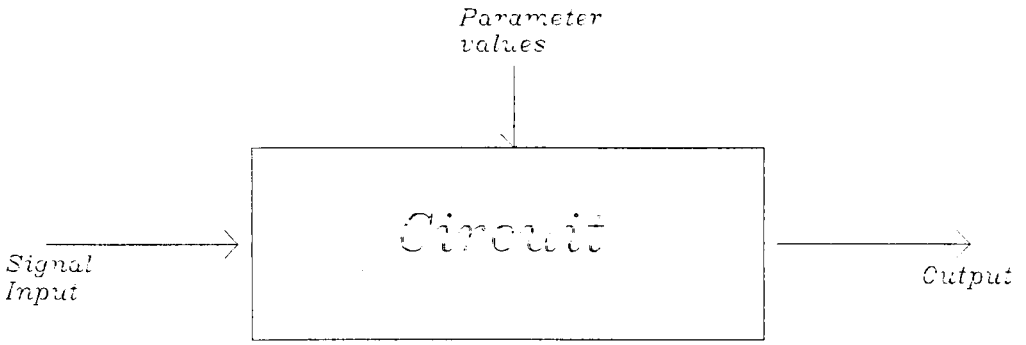


Figure 1.3: Circuit representation

emulate the environment in which that subsystem lives.

1.2 Critique of circuit optimization

1.2.1 Problem definition

This section is devoted to a review of current state-of-the-art circuit optimization techniques to provide the foundations for using electronic circuits in the RED case studies throughout this thesis. The methods used should be contrasted with the review of Robust Engineering Design methods which follows.

A necessary step in all optimization strategies is the evaluation of an objective function, the cost of evaluation being a vital factor for successful optimization. In circuit optimization this means measuring circuit response for varying values of input factors (signal input, temperature, component parameter values etc.). It is impractical to physically build and test an electronic circuit for each optimization step, so the circuit is approximated with a mathematical model. Electronic circuits can also be modelled with computer simulation packages such as SPICE [32].

An electronic circuit is represented with a 'black box' style arrangement (Figure 1.3). For Robust Engineering Design we wish to include the parameter values and the input signal values together as inputs parameters to an *empirical* model of the system. This is in contrast to mathematical models (e.g. state-space representation) which separate the signal input from the model parameters. The rationale for this is that we wish to model the system by observing its behaviour in its operational environment to see how variation in inputs translate to variation in system response. The inputs $X = (x_1, \ldots, x_d)$ are component values and signal inputs and the outputs $Y = (y_1, ..., y_n)$ are measurable circuit responses, consistent with the definition of quality in Section 1.1.1 we describe the action of a circuit as some function f where

$$Y = f(X) \tag{1.8}$$

We define input and output spaces as parameter space $\mathcal{R}_p$ and performance space $\mathcal{R}_q$ respectively. The function f can be thought of as the mapping function from one space to another. Figure 1.4 shows this for a system with two inputs and two outputs. The aim of modelling is to translate points from $\mathcal{R}_p$ to $\mathcal{R}_q$ and the following discussion of circuit optimisation is based on these terms. This generalisation into parameter and performance spaces will be used in the sense of empirical modelling of circuits, thus the parameter space includes model parameters and signal inputs.

1.2.2 Outline

We describe several approaches to the circuit optimization problem including basic nominal circuit optimization, Monte Carlo and Taguchi methods. The Section ends with

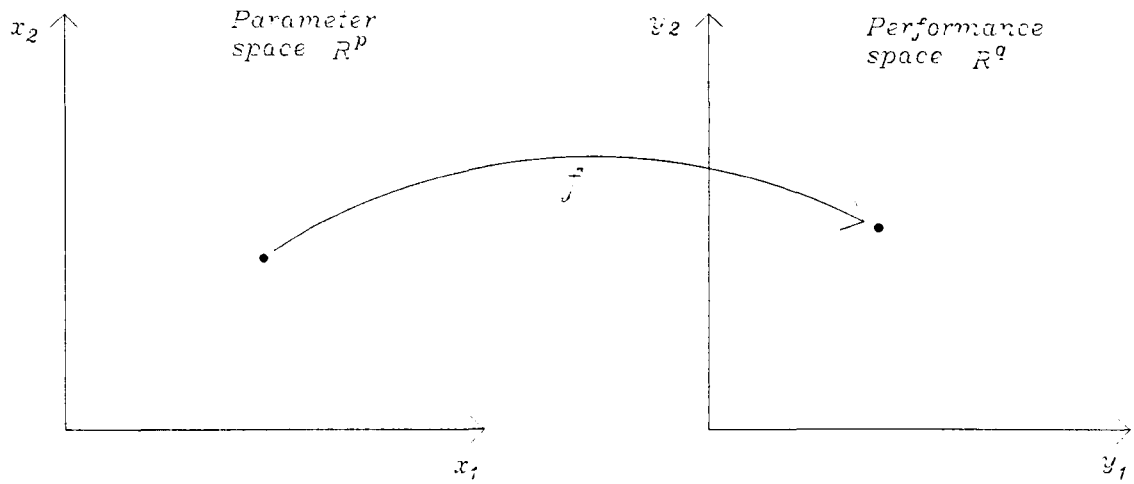


Figure 1.4: Mapping of parameter and performance spaces

a description of the more recent approach of Robust Engineering Design and its application to circuit design.

1.2.3 Classical Circuit Optimization

The basic goal in circuit optimization is to design circuits to meet some target set of output specifications Y_t, where $Y_t = (y_1, \ldots, y_n)$. Depending on the problem Y_t may be a single vector of values or may specify a range of acceptable values for each of n target specifications, defining an *acceptable region* $\mathcal{R}_y$.

Nominal design

The process of nominal circuit design uses numerical optimization to find a single set of circuit parameters X_0 which give a design with responses $Y_0 = Y_t$ (or more generally $Y_0 \in \mathcal{R}_y$). To achieve this the circuit, represented by the function f, can be modelled (e.g. with a simulator) by $\hat{f}$. This model is used in conjunction with an optimizer to find

a suitable set X^* for which $Y \in \mathcal{R}_y$. Nominal design focuses on how to obtain $\hat{f}$ and the selection of the most suitable optimization algorithm. There are many powerful gradient-based optimization algorithms, however deriving gradient information requires the use of special techniques [47] often involving matrix manipulations. Requirements for the exact gradients of all elements of Y with respect to all elements of X can therefore preclude their use in circuit optimization. Indeed the main problem in nominal design is approximating the gradients so the chosen optimization algorithm can adjust the X values to bring the Y values within $\mathcal{R}_y$.

Use of a circuit simulator

Circuit simulators may be used to translate points from parameter space to performance space. We represent a circuit with n responses $Y = (y_1, \ldots, y_n)$ and d input factors $X = (x_1, \ldots, x_d)$ as

$$Y = g(X), \tag{1.9}$$

the simulator acts as an approximation to the function $g(X)$ which is represented by

$$\hat{Y} = f(X). \tag{1.10}$$

The quality of the model depends on the type of simulator, the type of circuit and the type of analysis required to obtain the responses Y. All these factors influence the optimization process. The use of simulators is discussed fully in Chapter 2.

Definition of an emulator

A central idea of the Robust Design methods used in this thesis is the use of an *emulator* which models the circuit simulator (also described as a 'surrogate' in recent work by Yeşilyurt and Patera [51, 52]). Given the definition of a simulator in Equation 1.10 an emulator of the simulator is defined as

$$\hat{Y} = \hat{f}(X). \tag{1.11}$$

Optimizing nominal circuit designs

Local numerical optimization of circuits can be achieved by finding the gradients of the variables

$$\nabla f(X) = \begin{vmatrix} \frac{\partial f}{\partial x_1} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial f}{\partial x_d} \end{vmatrix} \tag{1.12}$$

within the system function. To compute these exactly can be costly so they are usually approximated. To improve the optimization process there has been work on improving these gradient estimations through the use of mathematical techniques [9] and further work has improved the efficiency of these methods [10].

Antreich et al. [6] describe the use of the SPICE circuit simulator as a basis for modelling a circuit, reducing the dimensionality of the resulting optimization problem by considering only those parameters for the circuit model which most affect the response.

In this case complexity is reduced by screening for important factors and eliminating others from the analysis by considering their sensitivities (Equation 1.12). This compares with principle component analysis (PCA) where most of the variability of a model is explained by linear combinations of components. Other work has concentrated on improving the optimization of the system model. Agnew [4] uses the minimax method of optimization on circuits following on from work by Charalambous and El-Turky [20].

Design for manufacture

In the nominal design approach circuit components are assigned particular parameter values. In reality circuit components are not manufactured at an exact value but are made to within a certain accuracy expressed as a nominal value with an associated tolerance which can be relative (e.g some percentage deviation e.g a resistor may be $10\text{K}\Omega \pm 10\%$) or absolute (e.g $10\text{K}\Omega \pm 100\Omega$) see Section 4.3 for a discussion of tolerances in optimization. Other effects outside the designer's control such as model uncertainties (especially for non-linear components such as transistors) and noise factors (temperature, humidity) will also affect the performance of the design. A circuit may therefore be optimized with respect to its nominal design but this represents only one of the many designs possible when considering mass production of the circuit. It may be the case that, after nominal optimization, when one examines one of a batch of circuits on a production line it will have a response Y lying outside $\mathcal{R}_y$.

The variability in X values and noise factors need to be taken into account in any assessment of design quality. This is because quality, as perceived by the customer, is related to how the design performs under manufacturing conditions and in the use

environment.

The problem of how to deal with unwanted variations in X can be addressed by considering the sensitivity of Y with respect to X. The goal is still the adjustment of X values to get Y within $\mathcal{R}_y$, as is the case for nominal design, the difference being that we want to do this while minimising the sensitivity of Y. This relates directly to the definition of quality in Section 1.1.1.

Schoeffler [36] deals with this problem by constructing differential equations using a sequence of equivalent networks which relate the circuit output to changes in each of its components. These equations enable the sensitivity of the circuit as a whole to be reduced with a suitable optimization algorithm. Director and Rohrer [22] derive sensitivity expressions for both linear and non-linear components by using Tellegens Theorem [46], also referred to in [17], to arrive at an equivalent circuit known as the Adjoint Network, this reduces the number of equivalent networks required to 1. The network is analysed and compared with the analysis of its 'adjoint' to arrive at sensitivity expressions for each variable with respect to the output parameters. Branin [16] derives sensitivity expressions for networks without reference to an equivalent network but using only matrix manipulation techniques, improving on the 'adjoint' approach by exploiting the fact that only one network simulation is required to produce sensitivity expressions. The use of Tellegen's theorem is also documented in [17]. It states that, for two different circuits having only circuit topology in common, the sum of the product of all voltages in one circuit with all currents in the equivalent branches of

the other circuit is zero. This can be expressed as

$$\sum_{all\ branches} i'_b v''_b = 0 \tag{1.13}$$

where i'_b is the the branch current of one circuit and v''_b is the corresponding branch voltage of the other circuit. We shall return to the exploitation of circuit topology for efficient analysis in Chapter 2.

Another approach to the design of component tolerances is to consider the effect that tolerancing has on the parameter and performance spaces of Figure 1.4.

The Performance Region

Expanding the concepts of parameter and performance space to include variation can provide information on the quality of the design. If we quantify variation by assigning lower and upper bounds on the vector $X = x_1, \ldots, x_m$ of the m input factors to give $X \pm \delta X$ then, instead of describing a point in parameter space, we describe an m-dimensional region $\mathcal{R}_x$ containing all possible combinations of parameter values for that design. The function f can then translate $\mathcal{R}_x$ to performance space $\mathcal{R}_q$ where the 'performance region', $\mathcal{R}'_x$ describes the variability of the circuit in the light of the variability described by $X \pm \delta X$. Figure 1.5 represents this action.

The problem is to locate the performance region $\mathcal{R}'_x$ and move it preferably inside $\mathcal{R}_y$, the region of acceptability. Once located, comparison of $\mathcal{R}'_x$ with $\mathcal{R}_y$ shows the design performance in the light of variation as in Figure 1.5.

Location of the performance region is attempted in several ways. Tahin and Spence [45] describe a method called the 'radial exploration' approach. This method approximates

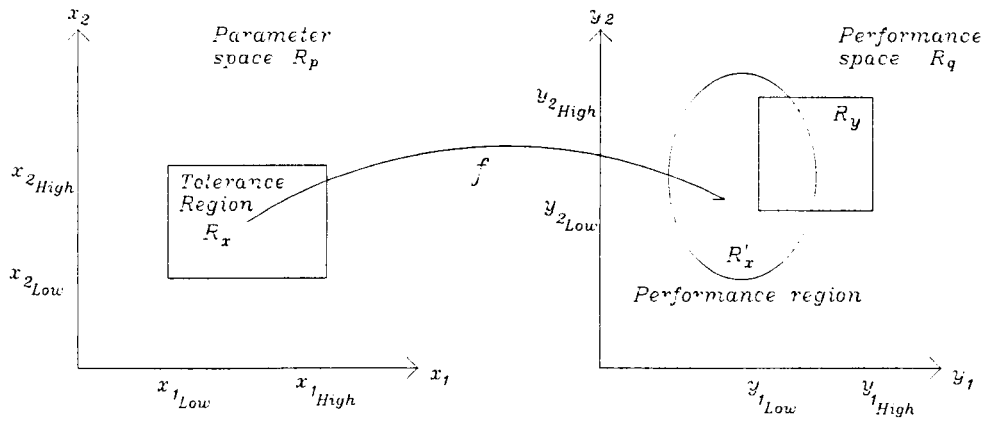


Figure 1.5: The general design scenario.

$\mathcal{R}'_x$ (feasible region) by searching radially from a point inside $\mathcal{R}'_x$ (e.g the nominal design) for its boundary. Points on the boundary are built up and the performance region is approximated from this. The technique uses mathematical programming to adjust the design parameters to do this.

Abdel-Malek [1] describes a geometrical method for approximating the performance region with an ellipsoid which decreases in volume to converge at the design centre. This is then applied to the technique of design centering. These techniques can be considered collectively as Inverse Engineering problems where one seeks to find a design solution given the performance specifications of a product. Set inversion falls into this category.

1.2.4 Set inversion

The basic idea of set inversion is to estimate the parameters of a function using interval arithmetic to translate between parameter and performance spaces. The process can be thought of as finding the inverse of the function f and is thus related to the concept of inverse engineering.

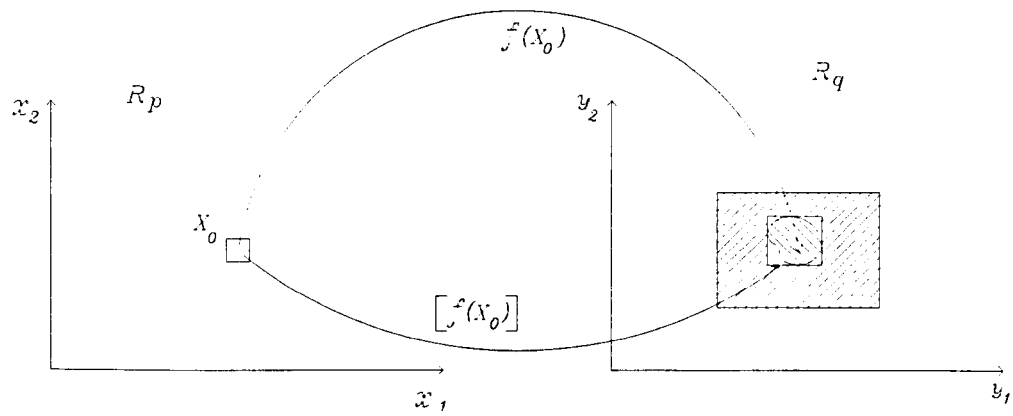


Figure 1.6: Translating intervals from $\mathcal{R}_p$ to $\mathcal{R}_q$.

Definitions. The process of parameter design through set inversion can also be visualised using the parameter space $\mathcal{R}_p$ and performance space $\mathcal{R}_q$. The parameter design problem is, given an acceptable region $\mathcal{R}_y$ in $\mathcal{R}_q$, to find the corresponding set of acceptable design parameters $\mathcal{R}_x$ in $\mathcal{R}_p$, to do this we need to find f^{-1}.

First steps. The basic concept involved in this approach is to use *interval analysis* to divide $\mathcal{R}_p$ into sub-spaces or boxes (n-dimensional intervals) and translate them one at a time to $\mathcal{R}_q$ as described in [27, 26]. Figure 1.6 shows this process for the set $X_0 = (x_1, x_2)$. These boxes can then be divided into three categories shown in Figure 1.7:

Infeasible box Where the set X_0 produces a response completely outside the acceptable region $\mathcal{R}_y$, case 1.

Undetermined box Where X_0 produces a response overlapping $\mathcal{R}_y$, case 2.

Feasible box Where X_0 produces a response inside $\mathcal{R}_y$, case 3.

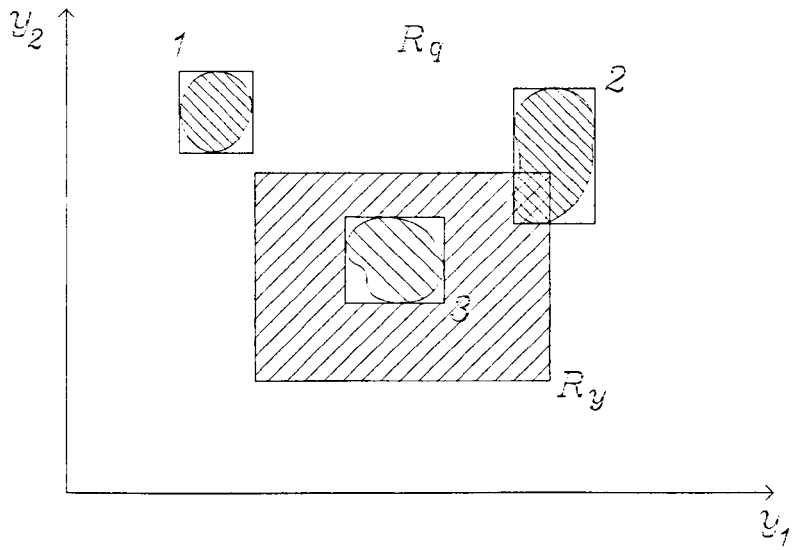


Figure 1.7: The three types of box.

Parameter estimation. Given the function f the tolerance region can thus be estimated. The size of interval used determines the accuracy of estimation of this area and an iterative procedure is used to improve estimation to within a given accuracy, this is termed bounded-error estimation. Figure 1.8 demonstrates the location of the tolerance region, the indeterminate set being unshaded.

Other approaches to the location of the tolerance region have been adopted using a mixture of geometrical techniques and circuit simulations. These are presented in the section on Monte Carlo methods.

1.2.5 Design Centering, Tolerancing and Yield Optimization

Design centering, tolerancing and yield optimization can be defined in terms of regions in performance space. The process of moving $\mathcal{R}'_x$ to within $\mathcal{R}_y$ is known as design centering. Adjusting the size of $\mathcal{R}'_x$ to fit inside $\mathcal{R}_y$ is called tolerancing. Yield

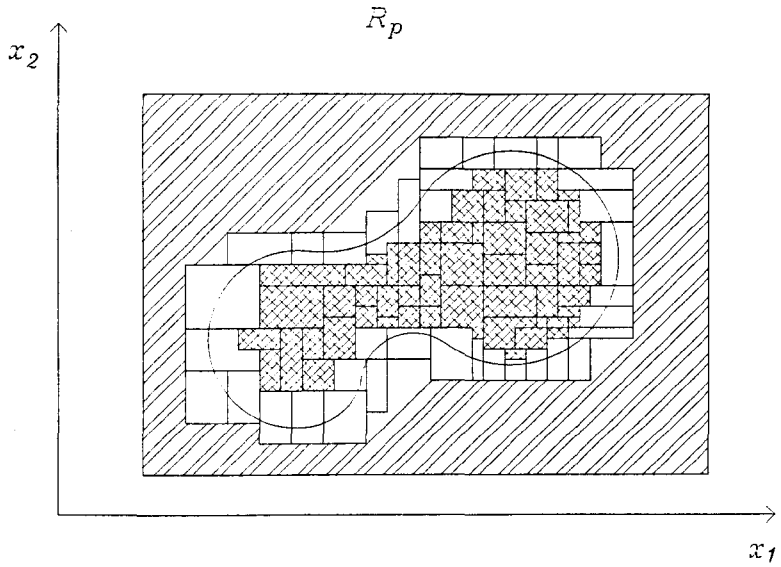


Figure 1.8: Estimation of tolerance region.

optimization is design centering, the difference being that it concentrates on how much the performance and acceptability regions overlap as a measure of manufacturing yield (see Figure 1.5).

Butler [19] defines the region of acceptability in terms of one dimensional sub-spaces. Each component is taken in turn (the others being held fixed) and its 'large change sensitivity' is calculated. This sensitivity is defined as how much from nominal the component can deviate before the circuit performance exceeds the specification. This concept is then expanded to produce 'performance contours' which are in effect second order sensitivities explaining how component sensitivities change with changing parameter values. These definitions are then used to desensitise and correctly tolerance an existing nominal design. Agnew [3] adopts a similar approach by defining a 'margin sensitivity' to allow algorithms to centre a design.

By combining the processes of nominal design and centering one can move the nominal

point inside the region of acceptability in such a way as to allow the largest set of component tolerances. This approach is presented by Bandler and Liu [11] and continued by Bandler, Liu and Tromp [13] and Bandler and Abdel-Malek [8]. In addition Bandler, Liu and Chen [12] have produced a computer package, TOLOPT, to implement this method.

In tackling the problem of yield optimization, Styblinski [41] along with Abdel-Malek and Bandler [2] begins by consideration of the probability density functions of the component distributions. That is how the probability of component values will vary across their defined ranges. The optimization, essentially a design centering process, involves a cost function to take into account the cost of circuits that do not meet the design specification.

1.2.6 Traditional Monte Carlo analysis

As an aid to the development of circuits that are insensitive to component tolerances, Monte Carlo analysis can provide information on how tolerances propagate through a circuit to affect the response Y. The basic idea is to vary randomly the parameter values within the tolerance range $X \pm \delta X$ and then to observe the effect on Y. The circuit is analysed many times varying X randomly and recording the Y's. This strategy is an attempt to translate not just a point in parameter space to performance space but the entire region defined by $X \pm \delta X$. Eventually, if enough simulations are performed, a cluster of points in the performance region will be obtained. Comparing the performance region with the region of acceptability allows the manufacturing yield of the circuit to be calculated by subtracting the designs lying outside the region of acceptability. This

information can be incorporated with optimization techniques to improve the design. Balaban and Golembeski [7] describe Monte Carlo methods applied to the design of practical circuits. Karafin [29] shows how Monte Carlo methods can be used to assign tolerances intelligently to a design. Butler [19] describes another method of design using Monte Carlo analysis. This uses 'large change sensitivity' as a measure of circuit performance where the 'large change sensitivity' of a component is how much its value can deviate from nominal before the design specification is exceeded. This allows the designer to desensitise a nominal design.

One problem is the mapping of X's to Y's. Computers can be used to simulate circuits to provide this mapping, however the simulation of large circuits can take many hours to complete depending on circuit complexity. We need to translate not just one set of inputs (i.e parameter values) from parameter to performance space but enough in order to estimate the performance region of the design well.

Improvements in Monte Carlo analysis

To obtain a good estimate of the performance region (necessary for optimization) requires many simulations. Because of this methods have been developed to reduce the number of analyses required for optimization using Monte Carlo methods.

Performance Region.

One of the main problems with Monte Carlo analysis is the time taken to get an estimate of the performance region. To improve on this several techniques have been employed which approximate the region using geometrical techniques along with fewer

solutions of the design equations. The 'simplicial approximation' method [21] approximates the performance region by finding points inside the region and using linear programming to interpolate the boundary thus allowing fewer analyses to give an approximation of the region. Tahin and Spence [45] compare the 'radial exploration approach' described earlier with a basic Monte Carlo method and show that the radial method is more efficient. Eckstein and Lüder [23] also reduce the number of simulations required within a Monte Carlo analysis by only sampling in areas which are most likely to contain acceptable circuits.

Monte Carlo iterations.

When employing certain optimization techniques it is necessary to perform repeated Monte Carlo analyses. Each time a design is improved a new Monte Carlo analysis is required for the circuit since it now has new parameter values. This is very costly in cpu time.

Research has been conducted on reducing the number of simulations required in such an iterative scheme. Parametric Sampling [38] uses a large pool of initial simulation results to home in on the acceptable region thus avoiding re-simulation of a particular design which could occur if two performance regions overlap. Stein [40] also reduces the number of simulations required by re-using old simulation results and only doing further sampling where the original sample distribution is undersampled. Soin and Spence [39] employ two methods (common points scheme and correlated sampling) to reduce the number of analyses required. These methods take advantage of any overlap that successive iterations may incur.

1.2.7 Conclusions

Despite the improvements in Monte Carlo analysis an accurate analysis of a large circuit can still take hours or even days to complete and this is a major drawback to the method. Because of the use of random designs the Monte Carlo method appears rather crude. A more efficient approach to the problem of optimizing a circuit is required to provide a more useful design tool.

1.3 Critique of Robust Engineering Design

1.3.1 Introduction

The aim of Robust Engineering Design (RED) is to produce systems robust against downstream variations in manufacture and use through a systematic design methodology. Systems are optimized using experimental results rather than the gradient calculations used in the methods outlined in Section 1.2.3. There are two general categories of RED strategy [37]; (i) the loss model (LM) approach, (ii) the response model (RM) approach. The difference between these two approaches lies in how the results of experimentation are used to optimize the design. In the LM approach the observed responses from the experiment are used directly to estimate the performance of the design whereas in the RM approach they are used to fit a model of the system which is used to predict the performance of the design as part of an optimization procedure. The general strategy is first to select a set of appropriate performance measures, $Y = (y_1, \ldots, y_t)$ and the set of input factors $X = (x_1, \ldots, x_d)$ possibly affecting Y. The system under observation can then be represented by $Y = f(X)$ and analysed at a

special set of test inputs, an *experimental design plan*, $\mathbf{X} = S_1, \ldots, S_n$ to produce values of Y_i at each S_i, $(i = 1, \ldots, n)$. Variability (including component tolerances) is introduced through careful selection of the experimental design plan. The type of plan used within RED is tailored according to the required method of estimating product performance. Section 1.3.2 and Section 2.3.1 describe different types of plan.
For the Response Model approach, the set (S_i, Y_i), $(i = 1, \ldots, n)$ is used to fit an empirical model to the system which is easier to compute than determining Y_i from the original system. This is termed an emulator, defined in Section 1.11. The emulator is then used to find an optimal setting X^* for the system parameters.

1.3.2 Loss Model - The Taguchi Method

The Loss Model approach estimates the 'loss' or 'risk' of a system (a criterion of the goodness of the system) directly from experimental observations. The most famous example of the Loss Model approach is the strategy introduced by Genichi Taguchi [44, 42] to improve the quality of products initially in Japan in the 60's and whose name subsequently became ubiquitous in international industry in the 80's. Taguchi describes an easily implementable strategy for improving product quality using this approach.

Experimental Design

To analyse a system the input factors $X = (x_1, \ldots, x_d)$ need separating into design factors $C = (c_1, \ldots, c_d)$ and noise factors $U = (u_1, \ldots, u_d)$. Design factors are parameters under control of the designer affecting Y. Noise factors are themselves split into two categories, internal noise U_{in} and external noise U_{ex}. Internal noise describes controllable variations such as component tolerances and manufacturing process variations. External

noise is uncontrollable e.g humidity, temperature etc. Setting $c_i = 0$ if the ith parameter is not a design factor and $u_i = 0$ if it is not a noise factor, allows us to write

$$X = C + U \tag{1.14}$$

Once defined the parameters form the basis for experimentation on the design to determine the performance characteristics of the circuit. The experimental design will provide the values of the system parameters to be used for a number of trials $X = S_i,\ (i = 1, \ldots, n)$ (either computer simulations or real hardware trials) of the given design. The results of this experiment are used to estimate the performance (or loss) of the circuit considering both internal and external noise.

The experimental design used for a Taguchi-style experiment is a product array formed from an inner array and an outer array. The outer array consists of rows $S_i^c,\ (i = 1, \ldots, n)$ where

$$S_i^c = c_i \otimes U_{in}. \tag{1.15}$$

where $\otimes$ is the cartesian (set) product. This represents the design factors C and the internal noise U_{in} i.e nominal values plus high and low settings. The external noise is represented by the inner array, $\mathbf{U_{ex}} = (U_{ex1}, \ldots, U_{exk})$ where k is the number of external noise factors. Each row of the outer array is modified by the inner array to mimic noise around the input parameters. The design is evaluated at the product of each row of the outer array with the inner array i.e

$$S_i^c \otimes \mathbf{U_{ex}},\ \ (i = 1, \ldots, n) \tag{1.16}$$

where again $\otimes$ is the cartesian product.

The number of input parameters increases with design size, this rapidly increases the number of trials needed in a product array experiment. To counter this effect fractional factorial arrays are used as design plans, this reduces the number of trials required by not taking all interactions between the x_i's. Taguchi typically uses Plackett-Burman type designs [33] which only estimate main effects: main effect orthogonal fractions.

Analysis

Central to the process of design is the definition of the quality of a product in terms of a 'loss function'. Taguchi defines quality as the characteristic that avoids loss to society once the product is shipped. This loss is measured in monetary terms. The loss function is then a way of uniting financial loss with deviation from functional specification. If we compare this idea with the definition of the acceptable region in parameter space, in which all designs are seen as good, a single point in that space represents the ideal design and any deviation from this point incurs a loss dictated by a loss function. The design method should seek to reduce this loss as much as possible given other constraints such as manufacturing cost.

A typical loss function for a Taguchi style approach is a quadratic, shown in figure 1.9, this defines the loss as increasing with the square of the distance of the real value obtained from the target value required. This is exactly the loss used in the definition of quality in Section 1.1.1 although Taguchi's use is more philosophical than mathematical. The concept of a region of acceptability is still valid, what has changed in effect is the importance of the response location within the region. One interpretation would be to

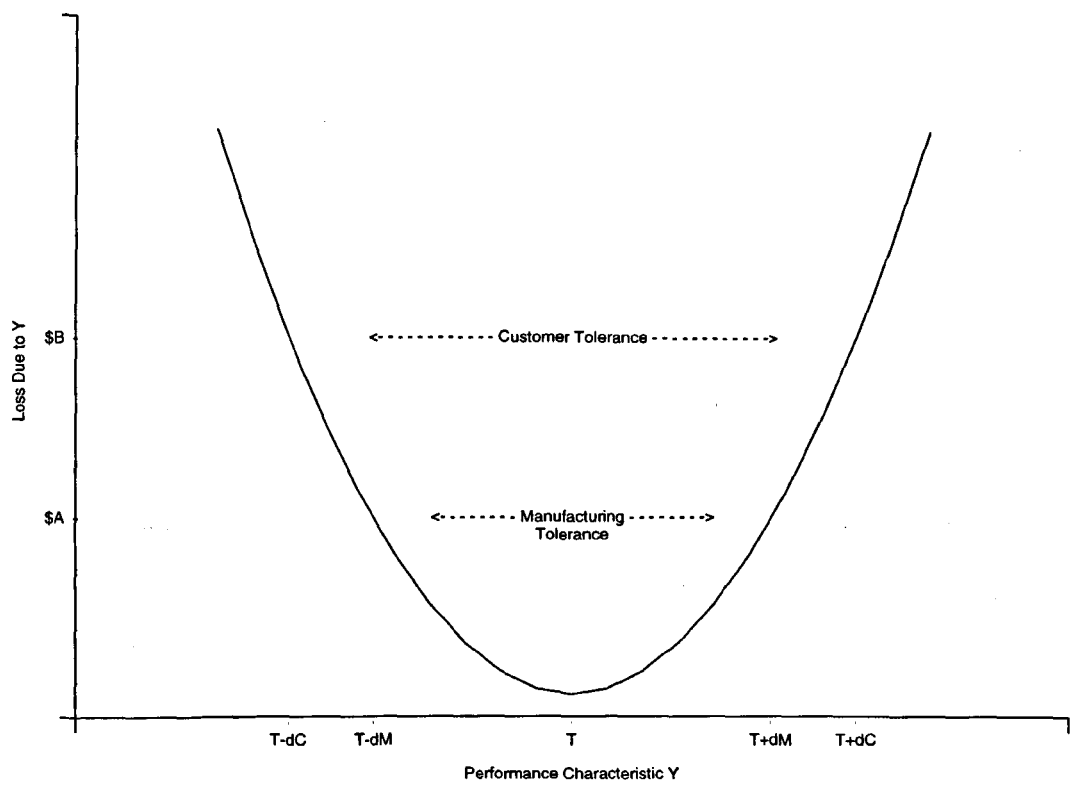


Figure 1.9: A typical loss model

require the distribution of responses within the region to be normal about the optimum response rather than uniform within the region of acceptability. The boundary of the region is still a useful concept when considering this, ideally it defines where the distribution of points falls to zero, this can be approximated by 3σ from the mean for a normal distribution, giving 100% yield.

In line with the loss function design performance is expressed in terms of signal-to-noise (SN) ratios, that is the ratio of the mean of the response (signal) to the variance (noise) for each y_i. With the SN ratios calculated for each trial of the experiment the dispersion and location factors can be identified and adjusted to bring the design to within $\mathcal{T}$. The dispersion factors are those x_i's which influence performance variability whereas location factors affect only the mean. The SN ratios defined by Taguchi (numbering over 60) need to be used carefully if they are to accurately represent the loss and have been criticised by Vining and Meyers [48] who present a more rigorous treatment of responses. Kackar [28] and Barker [14] both describe the methods employed by Taguchi. In addition Taguchi [43] describes the application of his methods to the analysis of an electrical circuit to demonstrate the capabilities of the approach.

An important contribution by Taguchi in the field of RED has been the dissemination of techniques in industry. Being an engineer by discipline Taguchi is able to present the statistical concepts of RED in a way accessible to engineers. There has recently been a lot of interest in Taguchi's approach to robust design and one of the main criticisms is that the techniques used are doubtful from a statistical inference point of view and moreover do not yield optimal solutions to the design problem.

1.3.3 The Response Model approach

There are two basic steps to optimization using the RM approach. First Y is estimated with a suitable model built from the results of a designed experiment. This model is then used as the objective function in a suitable optimizer to predict the performance of the design. The type of model used to estimate Y and the general approach in reducing variability in the design define different categories within the RM approach.

Response Surface Methodology (RSM) is one such category that uses regression models to estimate Y and generally attempts to minimise variance and adjust X to bring Y within $\mathcal{T}$. This is similar to the 'Dual Response approach' [48] also used in LM methods where the mean and variance of a response are used for optimization and is essentially equivalent to the 'unbiased' approximation of Equation 1.7.

In this thesis we adopt the RM method of DACE (Section 2.3.2) which uses a Gaussian stochastic process to estimate Y. The highly adaptive nature of this type of model makes it more suitable for use in modelling high-dimensional systems than the more basic polynomial model. The general RED procedure is outlined in Section 3.2.6.

Modelling Y directly eliminates the need for the Taguchi inner array used in the LM approach as it is less important to replicate experimental trials at the same design factor settings (i.e use the inner array to vary c_i's about their outer array values). In RM methods design and noise factors are varied together using a *combined* array as the experimental plan. This saves in the number of trials needed to conduct an experiment - an important consideration where time and resources are limited.

The major benefit of RM methods over LM methods and other techniques such as Monte Carlo is the fact that a model of the design is generated. This model, compared with

analysis of the original design, is fast to compute and can be used directly in a design optimization process. Low and Director [30] describe the process of modelling the response of an integrated circuit for design centering, Alvarez et al. [5] also demonstrate how RSM can aid in the design of VLSI devices. Yeşilyurt and Patera [52] also describe a method of modelling for optimization.

The use of DACE in optimizing more complex design situations is detailed in [15]. The more complex modelling strategy of DACE improves on the use of standard regression models in fitting a model to the design. The experimental design plan used for modelling with DACE is Latin Hypercube Sampling (LHS) which is shown to have good space-filling properties (i.e the values chosen for X using the LHS plan are well spread in parameter space). For a rigorous treatment of the statistical theory of experimental design and model building the reader is referred to [18].

1.4 Conclusions

Designing a product for both the manufacture and use environments needs the consideration of many factors. These factors include manufacturing processes and component cost and variability.

Mathematical (geometrical) techniques can reduce design variability in combination with numerical optimizers and the Monte Carlo method performs a similar function but both methods are costly in computer time.

The technique highlighted by Taguchi is an improvement, providing the design engineer with a framework to approach the problem of variability in design. The use of orthogonal arrays to reduce the time required for an analysis of the product design together with the

introduction of a noise array into the experimental design provides a more efficient design method than local sensitivity-based optimization or one-at-a-time experimentation. The interest generated in Robust Design by Taguchi has prompted renewed application of Response Modelling and statistical methods in general in the field of computer experimentation. An example of the recent efforts to improve the techniques of Robust Design is the Design and Analysis of Computer Experiments (DACE) [35] which describes the use of statistical methods in Robust Design to provide a more efficient design process.

References

[1] H L Abdel-Malek. The ellipsoidal technique for design centering and region approximation. *IEEE Trans. Computer Aided Design.*, 10:1006–1013, Aug 1991.

[2] H L Abdel-Malek and J W Bandler. Yield optimizations for arbitrary statistical distributions: Parts i & ii. *IEEE Trans. Circuits Syst.*, CAS-27:245–262, Apr 1980.

[3] D Agnew. Design centering and tolerancing via margin sensitivity minimization. *IEE Proc - G*, 127:270–277, Dec 1980.

[4] D Agnew. Improved minimax optimization for circuit design. *IEEE Trans. Circuits & Systs.*, CAS-28:791–803, Aug 1981.

[5] Antonio R Alvarez, Behrooz L Abdi, Dennis L Young, Harrison D Weed, Jim Teplik, and Eric R Herald. Application of statistical design and response surface methods to computer-aided VLSI device design. *IEEE Trans. Computer Aided Design*, 7:272–288, Feb 1988.

[6] K J Antreich, P Leibner, and F Pőrnbacher. Nominal design of integrated circuits on circuit level by an interactive improvement method. *IEEE Trans. Circuits & Systs.*, 35:1501–1511, Dec 1988.

[7] P Balaban and J J Golembeski. Statistical analysis for practical circuit design. *IEEE Trans. Circuits & Systs.*, CAS-22:101–109, Feb 1975.

[8] J W Bandler and H L Abdel-Malek. Optimal centering tolerancing and yield determination via updated approximations and cuts. *IEEE Trans. Circuits & Systs.*, CAS-25:853–871, Oct 1978.

[9] J W Bandler, S H Chen, S Daijavad, and K Madsen. Efficient gradient approximations for non-linear optimization of circuits and systems. *Proc. IEEE Int. Symp. Circuits Syst.*, pages 964–966, 1986.

[10] J W Bandler, S II Chen, S Daijavad, and K Madsen. Efficient optimization with integrated gradient approximations. *IEEE Trans. Microwave Theory Tech.*, MTT-36:444–455, Feb 1988.

[11] J W Bandler and P C Liu. Automated network design with optimal tolerances. *IEEE Trans. Circuits & Systs.*, CAS-21:219–222, March 1974.

[12] J W Bandler, P C Liu, and J H K Chen. Worst case network tolerance optimization. *IEEE Trans. Microwave Theory & Tech.*, MTT-20:630–641, Aug 1975.

[13] J W Bandler, P C Liu, and H Tromp. A nonlinear programming approach to optimal design centering tolerancing and tuning. *IEEE Trans. Circuits & Systs.*, CAS-23:155–165, March 1976.

[14] T B Barker. Quality engineering by design: Taguchi's philosophy. *Quality Assurance*, 13:72–80, Sept 1987.

[15] Maria C. Bernardo, Robert Buck, Lishin Liu, William A Nazaret, Jerome Sacks, and William J Welch. Integrated circuit design optimization using a sequential strategy. *IEEE Trans. Computer Aided Design.*, CAD-11:361–372, 1992.

[16] Franklin H Branin. Computer methods of network analysis. *Proceedings of the IEEE*, 55:1787–1801, Nov 1967.

[17] R K Brayton and R Spence. *Sensitivity and optimization.* Elsevier, Amsterdam, 1980.

[18] R J Buck. *The design and analysis of computer experiments.* PhD thesis, City University, London, UK, 1994.

[19] E M Butler. Realistic design using large change sensitivities and performance contours. *IEEE Trans. Circuit Theory*, CT-18:58–66, Jan 1971.

[20] C Charalambous and M El-Turky. Circuit design using a recent minimax approach. *Computer Aided Design*, 11(1):27–31, Jan 1979.

[21] S W Director and G D Hachtel. The simplicial approximation approach to design centering. *IEEE Trans. Circuits & Systs.*, CAS-24:363–372, July 1977.

[22] S W Director and R A Rohrer. The generalized adjoint network and network sensitivities. *IEEE Trans. Circuit Theory*, CT-16:318–323, Aug 1969.

[23] T Eckstein and E Luder. Design centering by improved monte carlo analysis of the region of acceptability. *Proc. IEEE Int. Symposium Circ. Syst.*, pages 951–954, 1986.

[24] N. Fenton and G. Hill. *Systems, construction and analysis.* McGraw-Hill, London, 1993.

[25] C M Fiduccia and R M Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181, 1982.

[26] L Jaulin and E Walter. Guaranteed nonlinear parameter-estimation from bounded-error data via interval-analysis. *Mathematics and Computers in Simulation*, 35(2):123–137, 1993.

[27] L Jaulin and E Walter. Set inversion via interval-analysis for nonlinear bounded-error estimation. *Automatica*, 29(4):1053–1064, 1993.

[28] R N Kackar. Off-line quality control parameter design and the taguchi method. *Journal Of Quality Technology*, 17:176–188, Oct 1985.

[29] B J Karafin. The optimum assignment of component tolerances for electrical networks. *Bell Syst. Tech. J*, 50:1225–1243, Apr 1971.

[30] K K Low and Stephen W Director. An efficient methodology for building macromodels of ic fabrication processes. *IEEE Trans. Computer Aided Design.*, CAD-8:1299–1313, Dec 1989.

[31] M. D. Morris, T. J. Mitchell, and D. Ylvisaker. Bayesian design and analysis of computer experiments: use of derivatives in surface prediction. *Technometrics*, 35(3):243–255, Aug 1993.

[32] L W Nagel. *SPICE 2. A computer program to simulate semiconductor circuits.* ERL Memo ERL-M520. Univ. California, Berkley, 1975.

[33] R. L. Plackett and J. P. Burman. The design of optimum multifactorial experiments. *Biometrika*, 33:305–325, 1946.

[34] J. Sacks, S. B. Schiller, and W. J. Welch. Designs for computer experiments. *Technometrics*, 31:41–47, 1989.

[35] Jerome Sacks, William J Welch, Toby J Mitchell, and Henry P Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–435, Nov 1989.

[36] J D Schoeffler. The synthesis of minimum sensitivity networks. *IEEE Trans. Circuit Theory*, CT-11:271–276, June 1964.

[37] Anne C. Shoemaker, Kwok-Leung Tsui, and C. F. Jeff Wu. Economical experimentation methods for robust design. *Technometrics*, 33:415–428, 1991.

[38] K Singhal and J F Pinel. Statistical design centering and tolerancing using parametric sampling. *IEEE Trans. Circuits Syst.*, CAS-28:692–701, July 1981.

[39] R S Soin and R Spence. Statistical exploration approach to design centering. *IEEE Proc-G*, 127:260–269, Dec 1980.

[40] M L Stein. An efficient method of sampling for statistical circuit design. *IEEE Trans. Computer Aided Design*, CAD-5:23–29, Jan 1986.

[41] M A Styblinski. Problems of yield gradient estimation for truncated probability density functions. *IEEE Trans. Computer Aided Design.*, CAD-5:30–38, Jan 1986.

[42] G Taguchi. Off-line and on-line quality control systems. In *ICQC*, pages B4.1–B4.5., Tokyo, 1978.

[43] G Taguchi. *System Of Experimental Design: engineering methods to optimize quality and minimize costs*, volume 1. Unipub / Kraus International Publications, White Plains, New York, 1987.

[44] G Taguchi and M S Phadke. Quality engineering through design optimization. In *IEEE Globe 1984 Conference Atlanta GA*, volume 3, pages 1106–1113, Nov 1984.

[45] K S Tahin and R Spence. A radial exploration approach to manufacturing yield estimation and design centering. *IEEE Trans. Circuits & Systs.*, CAS-26:768–774, Sept 1979.

[46] B D H Tellegen. A general network theorem with applications. *Proc. Inst. Radio Engineers, Australia*, 14:265–270, 1953.

[47] Gabor C Temes and Donald A Calahan. Computer-aided network optimization the state of the art. *IEEE Proceedings*, 55:1832–1863, Nov 1967.

[48] G Geoffrey Vining and Raymond H Myers. Combining taguchi and response surface philosophies : A dual response approach. *Journal of Quality Technology*, 22:38–44, Jan 1990.

[49] G W Wasilkowski and H Woźniakowski. There exists a linear problem with infinite combinatory complexity. *Journal of Complexity*, 9:326–337, 1993.

[50] W. J. Welch, R. J. Buck, J. Sacks, H. P. Wynn, T. J. Mitchell, and M. D. Morris. Screening, predicting, and computer experiments. *Technometrics*, 34(1):15–25, 1992.

[51] S Yeşilyurt, C K Ghaddar, M E Cruz, and A T Patera. Bayesian-validated surrogates for noisey computer simulations; application to random media. *SIAM journal on scientific computing*, To appear.

[52] S Yeşilyurt and A T Patera. Surrogates for numerical simulations; optimization of eddy-promoter heat exchangers. *Computer methods in applied mechanics and engineering*, 1994.

Chapter 2

Tools and techniques

2.1 Introduction

In this Chapter the methods used in the technical chapters of the thesis are described including

i. circuit simulation,

ii. experimental design and system modelling,

iii. heuristic decomposition algorithms and

iv. reviews of simulation and decomposition methods.

The thesis is concerned with the application of Robust Engineering Design methods to engineering systems. Each of the following Chapters describes different applications of RED, each application being highlighted by example studies in electronic circuit analysis. The topics of circuit simulation, experimental design, model-building and optimization are covered as stages in the RED process. Methods of combining system knowledge with

these stages lead to the use of decomposition algorithms and applications of sparse matrix techniques which are also discussed. There are also reviews of the simulation methods and decomposition techniques which are integrated with RED technology.

2.1.1 Computer simulation

The execution of real experiments for RED is sometimes impractical due to physical or economic constraints. Because of this RED may need to rely on computer simulation of the system under analysis using software such as SPICE [28] for electronic circuits or other Computer Aided Engineering packages such as solid body modellers. Performing computer experiments on systems for RED involves simulating the same system many times and this makes the time taken for each individual analysis important when considering the efficiency of the whole experiment.

Computer simulation of systems can bring benefits to RED in the following ways:

i. Computer simulation is generally faster than real experimentation.

ii. Repeated simulations with varying input conditions can be easily catered for.

iii. The RED process can be conducted 'off-line' without any intrusion into manufacturing.

iv. Simulation avoids building test prototypes for experimentation.

The RED process is dependent on the efficiency and accuracy of simulation method used because models are built from the results of computer simulations. Simulators therefore need to be fast, efficient and accurate to enhance the RED process. In this thesis the examples given use the well proven SPICE circuit simulator, described in Section 2.2.2.

2.1.2 System decomposition

As a system increases in size the number of possible combinations of parameters that significantly affect system response will increase combinatorially. To counter this effect a method of partitioning systems into sub-systems for analysis is needed. The effect of partitioning is to reduce the problem to a set of sub-problems which can then be analysed more efficiently.

2.1.3 Application to RED

The main hypotheses of the thesis is that RED can be improved by including techniques for reducing problem complexity. This Chapter describes tools and techniques from different disciplines which are employed throughout the thesis to reduce complexity.

2.2 Simulating circuits

For the work contained in this thesis the SPICE simulator is used to perform RED experiments. Section 2.6 comprises a thorough review of current research in simulation methods. The following sections are devoted to a basic description of SPICE in the context of performing RED experiments. Because RED involves the fitting of a mathematical model of the simulator, defined as an emulator in Section 1.11, the quality of the simulator plays a direct part in the accuracy of the emulation model. There is a strong move towards integrating analogue and digital simulation methods making simulation problems larger and thereby increasing the usefulness of fast, efficient simulators and methods of problem reduction through decomposition (see Chapter 5). Pederson [33] provides a good background to the development of simulation methods on

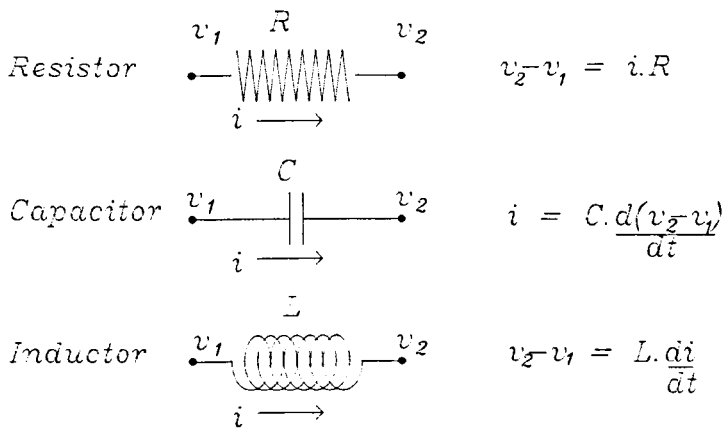


Figure 2.1: Three basic component types

which current development is based. This includes a discussion of the fundamental mathematical techniques used to simulate circuits such as nodal analysis and modified nodal analysis (SPICE2), large-scale techniques and timing and relaxation-based simulation (Section 2.6).

2.2.1 Mathematical modelling of circuits

Component modelling

The modelling of electronic circuits begins with the modelling of individual components which is central to the success of any simulation method. For two-terminal linear devices this is a straightforward process yielding up to second order differential equations relating current and voltage. The three basic component types are represented in Figure 2.1. Other ideal components such as switches, voltage and current sources etc. can also be defined. The equations in Figure 2.1 define the characteristics of these devices which are referred to as 'ideal' because of their simplified nature. Models more accurate than these ideal representations can be created by accounting for the non-linearities associated with

real components through the addition of more ideal components. An example of this is in the use of equivalent circuit models for non-linear device modelling (see below).

Device modelling

For more complicated non-linear components, such as transistors, physics-based mathematical equations which model the behaviour of the device are generally too complicated for analytical solution. These models need to be solved numerically at great computational cost. This has led to the use of 'equivalent' circuit models, based on combinations of ideal linear components, to approximate the behaviour of these non-linear devices. The parameters of 'equivalent' circuit models do not relate to physical device parameters. This effectively cuts the link between device manufacture and use for individual circuit designs because varying these parameters does not vary device performance in a realistic way. Including 'equivalent' circuit model parameters as part of an RED experiment will not then provide a direct link between circuit performance and device characteristics without understanding the relationship between the two sets of parameters. Establishing the link between device fabrication and use is critical if RED is to be applied to Integrated Circuits, Bandler, Biernacki, Cai, Chen, Ye and Zhang [2] describe the integration of physics-based models to circuit simulators for the purposes of design optimization, however the use of physics-based models is beyond the scope of this thesis (see Chapter 8).

Other methods of modelling devices for circuit simulation include

i. behavioural modelling ; where the device behaviour (usually a digital device) is encapsulated in computer code used by the simulator,

ii. hardware modelling ; where the device is physically connected to the simulator and incorporated directly in the simulation,

iii. macromodelling ; where the device characteristics are modelled mathematically,

iv. VHDL ; this is a standard language for representing digital circuits with computer code (see Section 2.6.3), and

v. HDL-A ; yet to be agreed on, this should provide a standard language for representing analogue circuits along the lines of (iv.) and help integrate analogue and digital circuit representation for mixed-mode simulation [14]. These modelling methods are commonly found in commercial packages.

Circuit modelling

The mathematical description of a circuit is achieved through the use of Kirchoff's current law (KCL), Kirchoff's voltage law (KVL) and the device (or branch) characteristics defined in the previous sections. These three laws combine to produce a set of equations to model the operation of the whole circuit. The KVL states that the sum of voltages in a closed loop around a circuit is zero, yielding a set of equations which can be represented as

$$AV = 0 \tag{2.1}$$

where A represents a $l \times b$ matrix of connections between b components (or branches) and l loops and V is a column vector of voltage drops $v_1, \ldots, v_b$ across the b branches.

Similarly the KCL yields the equation

$$BI = 0 \tag{2.2}$$

where B is a $n \times b$ matrix of connections between n nodes and b branches and I is a column vector of currents $i_1, \ldots, i_b$ flowing through each branch. The KVL and KCL equations provide topological information about the circuit and the addition of the branch characteristics connects these to fully describe the circuit.

This information will be exploited in Chapter 7 to reduce the complexity of RED experiments.

The use of impedance matrices derived from this approach for sensitivity analysis is referred to in Chapter 1. *Chapter 5 exploits this representation of system equations to partition circuits.* This approach is conceptually similar to that described in [6] where parts of a circuit's impedance matrix are suppressed during analysis to improve efficiency. A detailed account of the formulation of network equations can be found in [11].

These three sets of equations form the basis for uniquely defining the solution to the network. In the case of linear networks they can be solved using Gaussian elimination, for the non-linear case a numerical technique such as the Newton-Raphson algorithm must be used.

Given a mathematical circuit model the task of a circuit simulator is to evaluate the model, given a specific input or stimulus, at given time or frequency points depending on the type of analysis required.

2.2.2 SPICE

SPICE (Simulation Programme with Integrated Circuit Emphasis), introduced to the public domain in 1975, is the most common circuit simulation package today. It uses the Newton-Raphson matrix solution method with Gaussian elimination solving the system equations to determine the DC operating point (the quiescent point) of the circuit and repeats this for specified frequencies during a frequency domain analysis or uses numerical integration techniques for a time domain analysis. The basic steps in the simulator, for a time-domain analysis and given an initial DC solution, are summarised as follows:

i. Formulate coupled set of non-linear first order differential algebraic equations representing the circuit.

ii. Replace the time derivatives in step (i) with finite difference approximations.

iii. Solve the non-linear equations with Newton-Raphson.

iv. Increment the time point and repeat step (iii).

An up-to-date review of SPICE is given in [30].

2.3 Modelling circuits and systems

This section covers the methods used to conduct RED experiments. They include strategies for creating experimental design plans and statistical models of systems as well as a sequential plan for experimentation. The general methodology is commonly referred to as DACE (Design and Analysis of Computer Experiments) and is described in [8] and [5]. The sequential strategy is

i. Choose a suitable model to estimate the performance Y of the system.

ii. Design an experiment and select *input factor* nominal values and tolerances, collect the data from the simulation runs.

iii. Use the data to estimate the parameters of the statistical model chosen in (i), call this the *emulator* of the simulator.

iv. Analyse the model response $\hat{Y}$ via *main effects*, i.e effects from individual factors, and *interactions* between factors.

v. If the emulator is not accurate enough select a smaller region (i.e reduce the range of the nominal values of input factors) where the optimal response is likely to occur. Repeat steps (ii) to (v).

vi. When the emulator is accurate enough optimize $\hat{Y}$. Do a confirmatory simulation at the optimized input factor settings. Return to step (iv) if necessary.

The two key choices in this process are the experimental design plan and the emulator. For complex non-linear problems the traditional methods of using factorial designs and polynomial response surface methods (see section 1.3) have been replaced by better 'space-filling' codes (used in computer experiments) and more adaptive models. The latter approach is adopted here.

2.3.1 Experimental designs

There are two types of design used in the thesis, Latin Hypercube Sampling designs and Integer Lattices.

Latin Hypercube Sampling designs

Latin Hypercube Sampling (LHS) designs are good for filling space in high dimensions. They are also fast to compute because of their pseudo-random nature. These facts make them highly suited for use as experimental design plans for RED experiments in high dimensions, that is involving a lot of input factors. Normalising the input factors so that they all lie in the range $[0, 1]$, all possible combinations of d input factors will occur in the space $[0, 1]^d$. For an experiment with n runs an LHS design is constructed by dividing the interval $[0, 1]$ into n equally spaced values for each of the d factors and randomising them. Let $\mathbf{z} = [0, 1, ..., n-1]$, where n is the number of runs in the experimental plan. Then

$$s_j = \frac{\pi_j(\mathbf{z}) + 1/2}{n}, \quad j = 1, \ldots, d \tag{2.3}$$

is the j^{th} column of the experimental design S, where $\pi_1, \ldots, \pi_d$ are independent random permutations of $\mathbf{z}$. This algorithm places the design points in the centre of the randomly selected sections of a grid. An example design plan in the range $[-1, 1]^4$ with $d = 4$ and $n = 18$ is shown in Figure 2.2. The points in the graphs show the four factors, $x_1, \ldots, x_4$ plotted against each other to give an idea of the space-filling properties of the design.

Improved LHS designs

A Latin hypercube design with the design points more uniformly spaced can be chosen by measuring the variability of the number of design points in a randomly located subregion of the experimental design space. To give an example, suppose we have a

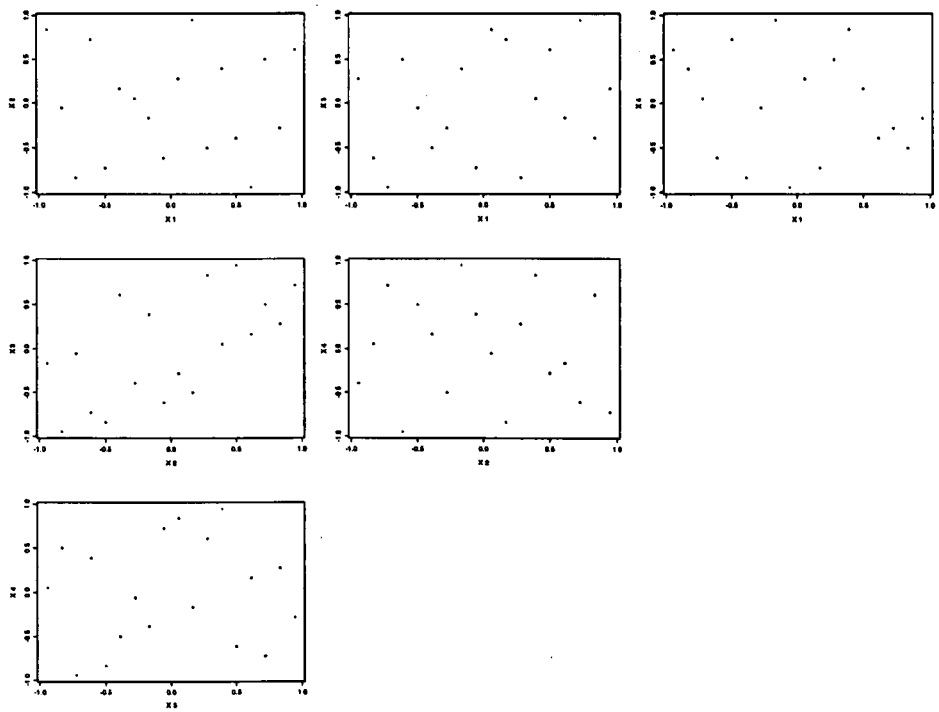


Figure 2.2: Plots of input factors for a LHS design plan

Latin hypercube design D. Then suppose a large number of rectangular subregions are placed in the design space and the number of design points in each subregion is counted. If the number of points in each region is the same or close to the same, then the points must be fairly evenly distributed in the design space. So if x_{ij} is the number of points in subregion j for the i^{th} design, then $Var(x_i)$ is the variance for design i and we select the design which minimizes $Var(x_i)$. To get an estimate of $Var(x_i)$ n_c randomly placed cubes are placed in the design space and $x_{ij}, j = 1, \ldots, n_r$ are used to estimate $Var(x_i)$. This is repeated for n_d designs and the design with the smallest value of $Var(x_i)$ is chosen to be the design. The number and size of the cubes and the number of designs to look at are chosen when the design is created.

Lattice designs

Lattice designs are another example of good space-filling designs which are easy to generate. The basic idea is to lay down points which are equally spaced along a trajectory given by a *generator* which 'wraps' around the input space, shown in Figure 2.3. A principal text is Niederreiter [31], and Fang and Wang ([38], and earlier work) make a considerable contribution in applications to statistics, including design. In a forthcoming book Zhigliavsky and Wynn [42] discuss applications to search and optimization. The one generator case is used here. Thus first select a sample size n and a single generator $(h_1, \ldots, h_d)$, where typically the h_i are integers. Points are generated in $[0,1]^d$ by taking successive multiples of the rescaled generator:

$$\left(\frac{kh_1}{n}, \ldots, \frac{kh_d}{n}\right) \mod(n); \quad (k = 1, \ldots, n)$$

where mod(n) means that the numerators kh_j are reduced mod(n). There are various good ways of choosing the h_j: (i) they should be primes or mutually prime to themselves and n (ii) they can be powers of a prime: $h_j = p^j$ where the powers are not equivalent mod(n) and n is prime (primitive roots). Designs can alternatively be chosen by pure optimization using an optimality criterion, see [4]. In the case study which follows we first choose n as a prime and then select the h_i generators according to (ii.) above. An example lattice in the range $[-1,1]^4$ with $d = 4$ and $n = 18$, as for the LHS design example, is shown in Figure 2.3.

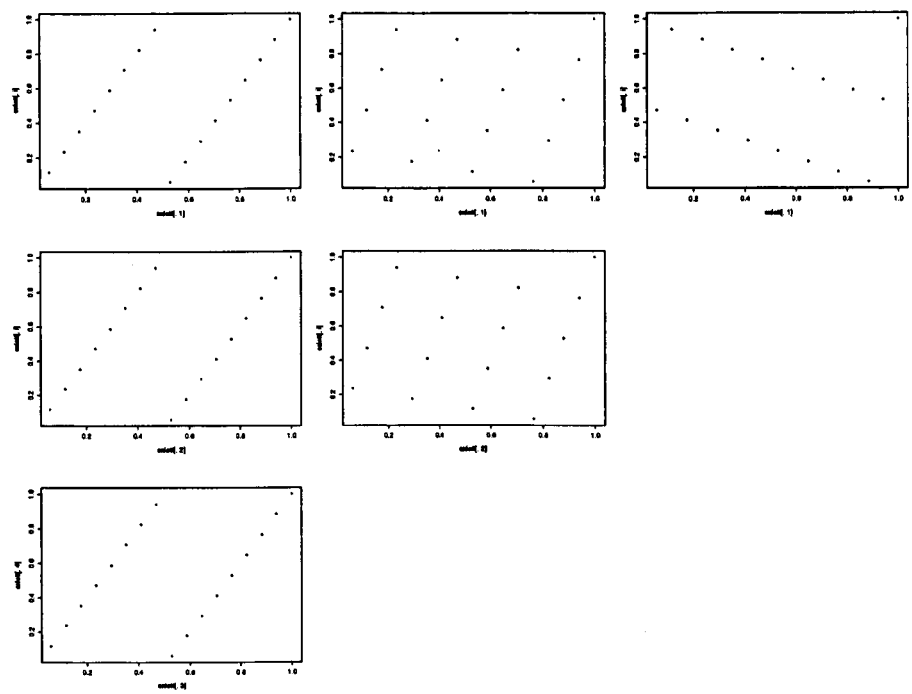


Figure 2.3: Plots of input factors for a lattice design plan

2.3.2 Statistical emulation

The response of a system with inputs and responses, or outputs, is emulated using a model with the independent variables being the input factors (circuit parameter values, signal inputs etc.) and the dependent variables being the system responses (frequency response, amplitude etc. for a circuit).

For the example cases in this thesis a statistical model is used to emulate a circuit simulator, this process is fully described in [34] and used in [5] to optimize the design of two IC circuits. The emulator is computed from data obtained by conducting a computer experiment according to an experimental design plan as described earlier in this section. In our case the model chosen includes only one regression term, β, which is

a constant. A brief description of the model follows. Consider the model

$$\mathrm{g}(\mathbf{x}) = \beta + \mathrm{Z}(\mathbf{x}) \tag{2.4}$$

where $\mathrm{Z}(\mathbf{x})$ is a random function and β is an unknown constant. At two sets of inputs, $\mathbf{x}$ and $\mathbf{x}'$, the covariance between $\mathrm{Z}(\mathbf{x})$ and $\mathrm{Z}(\mathbf{x}')$ is

$$Cov(\mathrm{Z}(\mathbf{x}), \mathrm{Z}(\mathbf{x}')) = \sigma^2 \mathbf{R}(\mathbf{x}, \mathbf{x}'). \tag{2.5}$$

The computer simulation of an electronic circuit design is conveniently represented by a realization of a random process. The philosophy is that although in reality there is no random error the stochastic process is a good way of summarising our ignorance of the behaviour of the output at unsampled inputs. The model can be used to predict the response of the same circuit under varying input conditions.

Let $\mathbf{g} = (g_1, \ldots, g_n)$ denote the observed performances at an experimental design of n input vectors, $\mathbf{s}_1, \ldots, \mathbf{s}_n$, and write

$$\mathbf{r}_\omega = [\mathbf{R}(\mathbf{x}, \mathbf{s}_k)] \qquad \mathbf{R}_\mathbf{s} = [\mathbf{R}(\mathbf{s}_k, \mathbf{s}_{k'})] \tag{2.6}$$

which are an $n \times 1$ vector and an $n \times n$ matrix, respectively. It can be shown (e.g. [34]) that the best linear unbiased predictor of $\mathrm{g}(\mathbf{x})$ at untried inputs of x when $\mathbf{R}$ is known is

$$\hat{\mathrm{g}}(\mathbf{x}) = \hat{\beta} + \mathbf{r}'_\omega \mathbf{R}_s^{-1}(\mathbf{g} - \hat{\beta}\mathbf{1}), \tag{2.7}$$

where

$$\hat{\beta} = (\mathbf{l}'\mathbf{R}_s^{-1}\mathbf{l})^{-1}\mathbf{l}'\mathbf{R}_s^{-1}\mathbf{g} \tag{2.8}$$

and $\mathbf{l}$ is a vector of 1's. For the examples we assume that $\mathbf{R}(\mathbf{x}, \mathbf{x}')$ is the family

$$\mathbf{R}(\mathbf{x}, \mathbf{x}') = \prod_i \exp(-\theta_i |\omega_i - \omega_i'|^{p_i}) \tag{2.9}$$

In applications the parameters θ_i and p_i are unknown and are estimated by maximum likelihood, but we omit the details, [8, 34, 5, 41], the emulator being constructed using a dedicated software package developed by R J Buck [7]. With this correlation structure two points, ω and ω', that are close together will have highly correlated g's. The predictor also has the exact interpolation property in that

$$\hat{\mathbf{g}}(\mathbf{s}_i) = \mathbf{g}(\mathbf{s}_i) \qquad\qquad i = 1, \ldots, n. \tag{2.10}$$

This property is typically *not* shared by traditional polynomial response surfaces.

2.4 Decomposition algorithms

A review of system decomposition is given in Section 2.7. In this thesis we use two distinct decomposition algorithms:

i. *partitioning*

ii. *tearing.*

These terms are defined in Section 1.1.4. The partitioning method uses an heuristic algorithm to physically decompose a system in two for analysis. The tearing method

uses an implementation of an algorithm to decompose a system by first representing its topology with an incidence matrix (see Section 2.4.2) and then using sparse matrix techniques to define sub-systems. The following two sections describe the algorithms used in this thesis to decompose circuits for analysis while Section 2.7 reviews system decomposition in general. Improvements to the algorithms are detailed in the chapters where they are used in order to preserve the general thesis format of separating other's work (here) from the technical chapters (Chapters 3 to 7). Both algorithms have been implemented in the C programming language and can be found in Appendix A.

2.4.1 Network partitioning algorithm

For network partitioning an improved implementation of the Fiduccia & Mattheyses [15] algorithm is used to partition a circuit, represented by a network graph, into separate sub-circuits. For the purposes of this algorithm a network is defined as a set of p cells $C = c_1, \ldots, c_p$ connected by a set of q nets $N = n_1, \ldots, n_q$. Given an initial partition $(\mathcal{A}, \mathcal{B})$ of the cells the algorithm moves a cell at a time from one block of the partition to the other in an attempt to minimize the the cutset of the final partition, the cutset being the set of nets connected to cells in both ($\mathcal{A}$ and $\mathcal{B}$) blocks; hence min-cut. After all moves have been made the best partition encountered during the pass is taken as the output. The algorithm can be repeated for a number of passes until no further improvement is made. Once a cell is moved it is locked in place for the remainder of that pass. A cell is selected for movement using two criteria:

Balance ratio. The balance ratio is defined as $r = |\mathcal{A}|/(|\mathcal{A}| + |\mathcal{B}|), 0 < r < 1$. Setting lower and upper bounds for this limits the number of cells in any one

partition to prevent the algorithm from the trivial solution of placing all cells in one partition (achieving a zero cutset).

Cell gain. For any partition $(\mathcal{A}, \mathcal{B})$ the gain g_i of cell c_i is the number of nets by which the cutset would decrease were c_i to be moved.

The main feature of the Fiduccia & Mattheyses (FM) algorithm is that it finds a good solution in linear time with respect to network size. This is achieved through the use of tailored data structures enabling cell selection and cell gain adjustment to be handled efficiently. The cells from each partition are placed in separate *bucket* arrays in order of their present cell gain. A *free cell list* contains the list of cells not yet moved for the current pass. The algorithm due to FM is thus:

i. Consider the first cell (if any) of highest gain from each block's *bucket* array, rejecting it if moving it would violate the condition on the balance ratio. If neither block has a qualifying cell, no more moves will be made.

ii. Among those cells returned in step (i), choose a cell of highest gain, breaking ties by choosing the one which gives the most even balance. Break remaining ties as desired.

iii. Return this as the *base cell*, c^b, remove it from its *bucket* array and place it on the *free cell list*.

The chosen cell is then moved, locked, and the effects on both net distribution and gains of neighbouring cells calculated to update the data structures. Achieving this in linear time requires care and uses the notion of a *critical net* defined as a net on which exists a cell that, if moved, would change the nets' cutstate. Given a partition $(\mathcal{A}, \mathcal{B})$, the

distribution of the cells on a net n_i is defined as the integer pair $(\mathcal{A}(n_i), \mathcal{B}(n_i))$ representing the number of cells on net n_i in blocks $\mathcal{A}$ and $\mathcal{B}$ respectively. A net n_i is defined as critical only if either ($\mathcal{A}(n_i)$ or $\mathcal{B}(n_i)$) equals 0 or 1. From this it can be shown that the gain of a cell depends only on its critical nets and that if a net is not critical before or after a move then it does not influence the gain of any of its cells. These observations allow the algorithm to compute passes in linear time as shown in [15].

2.4.2 Network tearing algorithm

Representing an electronic circuit as a graph $\mathcal{G}(X; E)$ with a set of nodes $X = x_1, \ldots, x_m$ and edges $E = e_1, \ldots, e_p$ we can relate the edges E with circuit components and the nodes X with circuit nodes. The graph $\mathcal{G}$ produces an incidence matrix of size $m \times m$ with $2p$ entries (note: the number of entries is $2p$ because the matrix is symmetric about the main diagonal). This matrix is analogous to the incidence matrix formed during the initial stages of a nodal analysis for circuit simulation [11]. The algorithm decomposes the graph $\mathcal{G}$ to produce an incidence matrix in a bordered-block diagonal (BBD) form with balanced block sizes and a minimally sized border. Figure 2.4 and Table 2.1 show an example graph with the resulting BBD incidence matrix produced using the algorithm described.

The following section describes a recently published algorithm [44] to decompose a sparse matrix into a bordered-block diagonal form for the purpose of tearing a system into sub-systems with a connecting network between them (see Section 2.7.3). The algorithm forms the starting point for modelling the system using a decomposition technique.

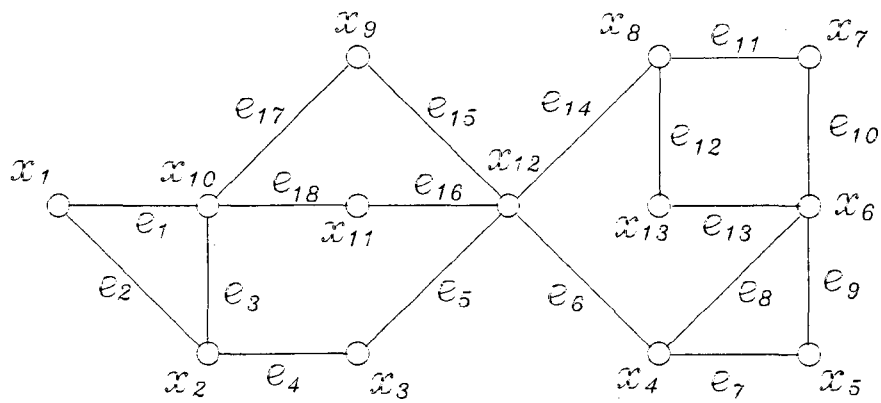


Figure 2.4: Example graph

*	0	e1	e3	e17	e18	0	0	0	0	0	0	0
0	*	0	e4	0	0	0	0	0	0	0	0	e5
e1	0	*	e2	0	0	0	0	0	0	0	0	0
e3	e4	e2	*	0	0	0	0	0	0	0	0	0
e17	0	0	0	*	0	0	0	0	0	0	0	e15
e18	0	0	0	0	*	0	0	0	0	0	0	e16
0	0	0	0	0	0	*	e9	e8	0	e10	e13	0
0	0	0	0	0	0	e9	*	e7	0	0	0	0
0	0	0	0	0	0	e8	e7	*	0	0	0	e6
0	0	0	0	0	0	0	0	0	*	e11	e12	e14
0	0	0	0	0	0	e10	0	0	e11	*	0	0
0	0	0	0	0	0	e13	0	0	e12	0	*	0
0	e5	0	0	e15	e16	0	0	e6	e14	0	0	*

Table 2.1: Matrix of example graph after decomposition

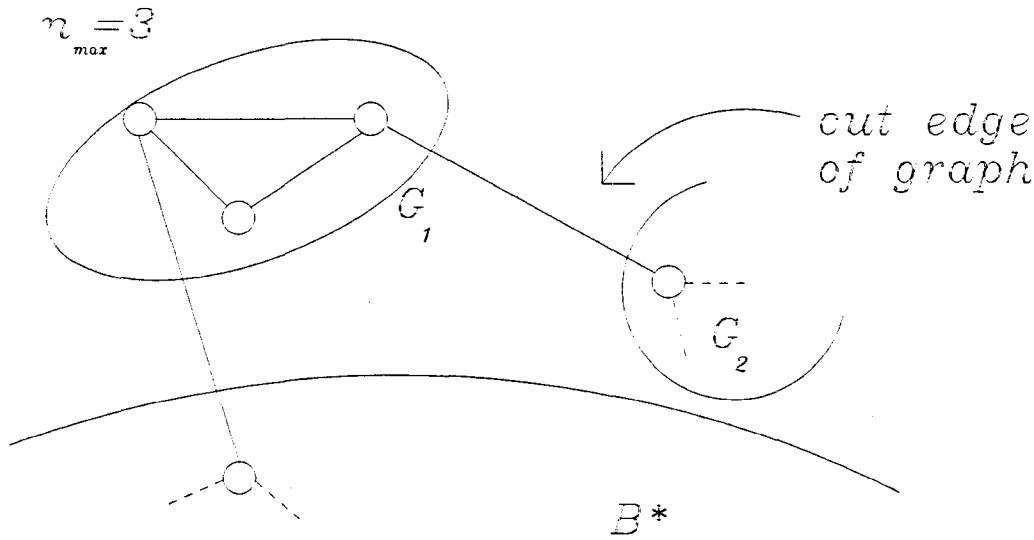


Figure 2.5: Part of graph showing a cut edge during group formation

The algorithm

The problem of finding a solution to the network partitioning problem is NP-complete (see Section 2.7.2) which means that no polynomial-time algorithm exists to find the exact solution to the problem. We therefore need to use a heuristic algorithm to obtain a solution to the problem in linear time.

Initialising the algorithm needs two variables to be defined, these are d_{min} and n_{max}. In its original form the algorithm seeks to partition the vertices X of a graph into a border group B and n other groups $G_1 \ldots G_n$ with the requirement that no group G_i is larger than the border group B. This gives a well balanced decomposition with similar sizes for B and all G_i's. For our purposes we require the border to be as small as possible requiring an enhancement to the algorithm adding the variable g_{max}. This extra variable defines the maximum size allowed for the G_i's and helps the algorithm minimise the size of B whilst fulfilling the requirement of maintaining balance among the G_i's. There are

two phases to the algorithm:

i. Initialization

   a. Construct an initial border set B from the set of graph vertices X where

$$B = x_i \in X : deg\ x_i \geq d_{min} \tag{2.11}$$

   b. Given B form groups G_i with the remaining vertices ensuring the size of $G_i \leq n_{max}$. If this condition is violated label the associated edge as cut. See Figure 2.5 for an example.

   c. Remove any cut edges formed in 2 by adding the necessary vertices to B forming $B*$.

ii. Border reduction Vertices are chosen one at a time to be moved from $B*$ to a connected group G_i choosing the vertex x_i connected to

   a. the least number of groups, or in the event of a tie

   b. the least number of other vertices in $B*$.

   This process is repeated until the size of the largest group $G_i > g_{max}$.

2.5 Optimization

Throughout the thesis we use an in-house optimizer based on the global branch and probability bound method from the work of Professor A. Zhigljavsky [45] and written under his direction. The global optimization routine used is one of a family of global random search algorithms. This algorithm is based on alternating between a global step

which selects random test points globally and steps which randomly select local points. The full details are described in [45]. Design optimization is discussed in Chapter 4.

2.6 Review of circuit simulation

The SPICE package is a good all-round simulator which has a large library of non-linear device models available and can simulate a large class of circuits without convergence problems. The price of this flexibility is computational efficiency, with the growing size of circuit designs comes the need to simulate larger circuits which can take an unacceptably long time to do using SPICE. This has led to the development of faster and more efficient methods through improvements in

i. sparse matrix techniques

ii. latency exploitation

iii. numerical integration techniques

iv. mathematical modelling

to reduce the complexity of simulation. A discussion of these developments, detailed in a review by Hachtel and Sangiovanni-Vincetelli [19], along with the latest developments in state-of-the-art simulation follows.

The ability to simulate large systems has a direct bearing on the measurement of the complexity of a design problem and the ability to design robustly. This section reviews the development of analogue simulation methods and their relationship with the requirements of Robust Engineering Design (RED). The review forms the majority of the section, with the last part devoted to looking at the special requirements of RED

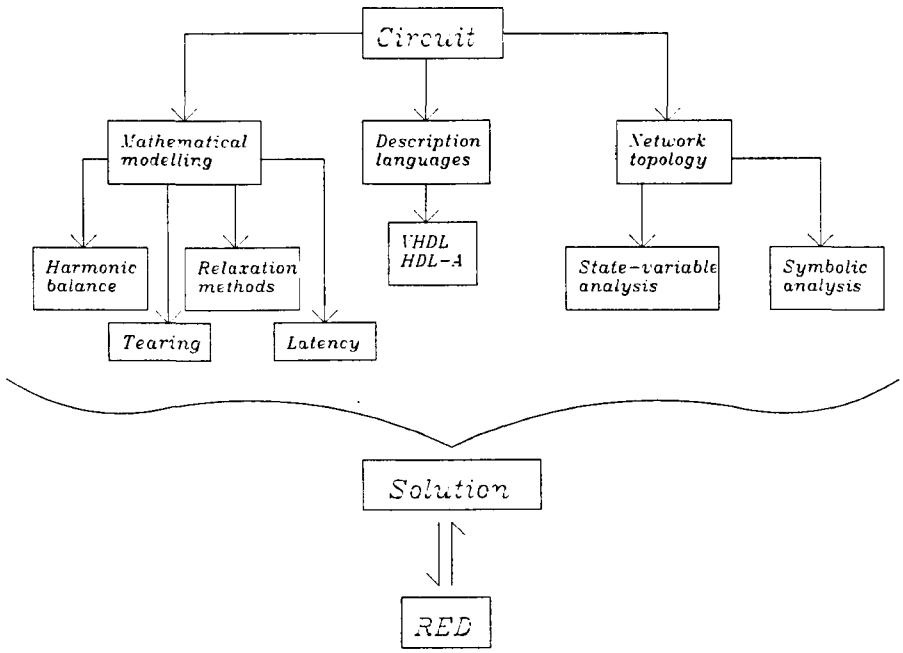


Figure 2.6: Overview of simulation types

and how these can be exploited to further improve simulator efficiency. Figure 2.6 shows how the different simulator types are connected and acts as a guide to the review in this section.

2.6.1 Third generation methods

This section is a review of simulation after the development of SPICE collectively referred to as 'Third generation methods' by Hachtel and Sangiovanni-Vincetelli [19].

Tearing

A way of reducing the complexity of simulation is by decomposing the problem of solving the matrix equations. First introduced as Diakoptics by Kron [25] the basic idea is to partition the system into sub-systems with an outer 'connecting' network linking the

two. The system equations can then be formulated for each sub-network and then linked together by the equations of the connecting network. The act of partitioning creates smaller sub-system matrices which can be equated more efficiently than the full system matrix.

It should be noted that Diakoptics involves the inversion of matrices which precludes the use of sparse matrix techniques; Tearing on the other hand uses a different mathematical approach allowing their use. By re-ordering the matrix equations so that the matrix is in a blocked form the system of equations can be partitioned in a different way. A classification due to [19] puts the different tearing methods into categories according to how the matrix is blocked for example: Bordered Block Diagonal (BBD), Bordered Block Triangular (BBT), Bordered Lower Triangular (BLT) etc.

The method of tearing does not separate the system variables or feedback paths associated with a system, however the overall effect of tearing is to reduce the computational complexity of the problem. The sparsity inherent in the system matrix is considered in [20], methods of finding suitable partitions are considered in [22] using numerical optimization and [35] using a heuristic approach.

Relaxation based methods and latency

Another way of simulating a circuit is to use relaxation techniques which replace numerical integration as the means of solving the system equations using an iterative process converging to a solution. Waveform relaxation is concerned with solving systems of differential equations while time-point relaxation is used to solve non-linear systems for specific points in time. The use of relaxation is described in [29] their advantages lie

in the reduced computation required for solution and their ability to exploit 'latency' in the system to improve efficiency when simulating. Latency occurs when, for a given time step, if a part of the design is not active (i.e does not move from its quiescent state) then its effect on the rest of the design is considered to be minimal and it is therefore not simulated for that step, this tends to happen more in digital circuits. The convergence of block (i.e using Tearing) relaxation methods related to circuit topology is discussed in [13] which describes sufficient conditions of the circuit topology required for the relaxation algorithm to converge to solution.

Harmonic balance

Harmonic balance (HB) is a mathematical technique used in simulation to find the steady-state solution to circuits with a periodic signal input (expressed by a Fourier series expansion), it is therefore used to simulate circuits in the frequency domain. The HB method converts the differential equations describing the system into a set of algebraic equations which can be solved using methods including numerical optimization, relaxation and Newton's method [26].

Compared with SPICE-style time domain analysis the HB method is a very efficient method for finding the steady-state solution for circuits, especially those which take a long time to settle (e.g high-Q circuits) because the transient response does not need to be calculated to obtain the solution.

2.6.2 Symbolic Analysis and simulation

Symbolic Analysis is concerned with finding the transfer function of a given electrical circuit, primarily in the frequency domain, in terms of variables instead of numerical

values. Thus instead of calculating the numerical solution of a large number of differential equations at every timestep the method formulates the equations (usually with the Laplacian operator S in the frequency domain) with symbols to produce a 'transfer function'. This 'transfer function' relates the output to the input and so gives the designer information on how individual components affect the system. The production of the symbolic transfer function takes more time than a numerical analysis but needs to be performed only once, simulation is then a matter of solving this one equation the required number of times. Compared with the numerical technique of solving the whole system of equations over and over the symbolic method is much faster. Lin [27] presents a survey of the techniques, which involve a topological analysis of the circuit, used in formulating symbolic functions. These can be summarized as:

i. tree enumeration

ii. signal flow graph

iii. state-variable analysis

iv. iterative method

v. nodal and eigenvalue method

Lin also gives example applications. A computer implementation of Symbolic Analysis for both analogue and digital circuits is presented by Singhal & Vlach [36]. The equations are formed with the Laplacian operator s, they therefore relate directly to the frequency response (where $s = j\omega$, ω = natural frequency, $j = \sqrt{-1}$). For time domain calculations these equations need to be inverted.

A major problem with the symbolic approach is that the size of the transfer function increases rapidly with the number of components in the circuit. The transfer function therefore takes a lot of computer time to produce and quickly becomes very large with a lot of terms. A full Symbolic Analysis circuit simulator is given by Gielgen, Walscharts and Sansen [17] which utilizes two techniques above those presented in [36] to improve efficiency. Firstly information about the characteristics of the circuit devices is exploited to produce simpler formulae: knowledge such as which are the largest/most important factors is given to the simulator, this allows the reduction of terms at the expense of model accuracy. Secondly the form of the calculated terms allows the calculation of second order effects in the circuit: this can aid the designer or an optimization routine in the formulation of a more robust design. Comparing the efficiency of the method to SPICE shows similar accuracy of results for an improvement in CPU time. The inclusion of an optimizer in a symbolic analysis package is discussed in [18], the symbolic functions are passed to an optimization routine (simulated annealing) and the best values for them are obtained given some quality criteria (see Section 1.1.1 for a definition of quality). The use of symbolic functions lends itself to optimization that is much faster than computing a full numerical analysis at every step and also has the advantage of being related to the topology of the design allowing more insight into the relationship between design and performance than numerical analysis.

2.6.3 Hardware description languages

Mainly used in digital simulation the VHDL language describes the functionality of a digital circuit in a format similar to high-level computer code. This avoids explicit

mathematical definition of the functionality of the circuit and is therefore more user friendly. The simulation is then a case of running the compiled code on computer. HDL's are currently being extended to analogue simulation in a similar way which paves the way for mixed-mode simulation (analogue and digital) in tandem with the integration of traditional style analogue and digital simulators which are currently being used.

2.7 Review of decomposition methods

This section outlines the general methods used for the decomposition and analysis of complex systems. Decomposition methods of partitioning are used to divide a system ready for analysis, these strategies include heuristics, clustering and optimization. Methods for the analysis of decomposed systems include diakoptics, direct decomposition and hierarchical decomposition. The decomposition techniques described are compared and evaluated with respect to the analysis techniques available. Decomposition and analysis are also related to the problem of the robust design of complex systems and criteria are given for the use of decomposition within a robust engineering design framework.

2.7.1 Introduction

When faced with a problem too large to be dealt with quickly or too complicated to have an obvious solution a natural approach is to break it up into several smaller tasks. Decomposition is concerned with the formulation of these tasks and, once defined, analysis of the resulting hierarchy in an attempt to reduce problem complexity. This has a particular use in RED where the combinatorial explosion encountered when dealing

with multiple inputs and outputs increases the complexity of the design and analysis of RED experiments.

When a large problem is solved through decomposition into sub-problems, there are several issues involved in finding a good solution:

i. quantifying the degree of difficulty of the problem

ii. method of decomposing the problem

iii. solving the sub-problems

iv. recombining the sub-problems

v. dealing with interactions between sub-problems

vi. testing whether solution of sub-problems guarantees solution of the overall problem.

These issues are inter-dependent to some extent, for example the method of decomposition usually defines how the sub-problems are solved and recombined.

The goal of system decomposition techniques is to enable the analysis of systems too complex to be tackled as a whole given the available tools and time. It is therefore useful to define complexity in terms of the resources, e.g computer speed or memory capacity, available to tackle the problem (see Section 1.1.3). For large systems "the curse of dimensionality", where complexity rises exponentially with problem size, means that any practical analysis involves the use of heuristics and/or decomposition methods to reduce complexity by taking care of the combinatorial explosion associated with handling a large number of variables and all possible interactions between them.

As noted in Section 1.1.4 there is a distinction between decomposition in a physical sense or *partitioning*, that is breaking up of a graph or network representing a system, and

decomposition at the mathematical level, *tearing*, where features such as matrix sparsity are used to decompose system equations into blocks for analysis. The key difference is in the formulation of system equations. Partitioning a network implies that no system equations have yet been formed and that separate sets of equations will result from the process (this is potentially useful for complex systems where formulating system equations could prove costly). On the other hand tearing requires the system equations to be stated for the full system before any partitioning, such as formulation of a sparse matrix in block form, can take place.

2.7.2 Partitioning

Methods exist for the partitioning of graphs used to represent systems. These methods can be adapted to produce partitions in a useful form in an attempt to reduce complexity. In this section several methods of partitioning are discussed including

i. The use of heuristics to minimize the number of connections between partitions. This finds locally optimal partitions in a practical time frame and divides the network into parts according to a given criterion such as finding a partition to minimise the number of connections between sub-graphs, usually improving on an initial, possibly random, partition.

ii. Clustering, concerned with grouping like objects to form partitions from scratch.

iii. Numerical optimization methods which can be employed directly in decomposition by defining the problem in terms of an objective function whose argument is the graph decomposition and whose value is some measure of the goodness of the partition. The optimizer then searches for a decomposition which optimizes that

function, either by improving on an initial partition, or starting from scratch.

It should be stressed again that, in terms of the analysis of large systems, these methods are applied directly to the physical system via a graph rather than the system equations.

Heuristic algorithms

Network Partitioning seeks to split a network, or graph, representing a system into distinct parts according to some specified metric. A major use of this technique is in VLSI chip layout where components need to be grouped so as to minimize the number of interconnections between them[24]. This metric, called 'min-cut' because we want to minimize the 'cut-set' of the graph partition, forms the basis for much work in this area[24, 43, 15, 10].

Mathematically the problem of finding the (globally) optimal min-cut for a network belongs to a class of problems which are NP-Complete [35, 16]. This means that no polynomial time solution exists and the time taken to find the global optimum will rise exponentially with circuit size. To deal with this problem an heuristic algorithm is usually employed to find at least a locally optimum solution to the min-cut problem. A notable contribution in this area came from Fiduccia and Mattheyses[15] who developed an algorithm for network partitions whose computation time grows, in the worst case, linearly with network size.

The partitioning of networks using heuristics generally concentrates on improving a given partition (refinement algorithms) rather than creating a partition from a network description. Heuristics are used to choose a cell to move from one block to another or exchange cells between blocks to improve the partition. In [15] the concept of cell gain is

used to select the 'base' cell (cell to move), this algorithm is described in Section 2.4.1. The defining heuristic of the algorithm is how the base cell is chosen. For large networks there may be more than one cell with the same gain competing for the position of base cell. Kernighan and Lin [24] expand the cell gain concept to improve base cell choice by looking one step ahead in the algorithm. This is referred to as 'second order gain'. Another improvement suggested by Kernighan & Lin takes the best solution from the algorithm, rearranges it and feeds it back in for another pass. Instead of starting the search from a random or arbitrary partition, the algorithm uses information gained from the most recent pass to select a new starting point. This provides a wider search of the solution space and increases the chances of finding a 'good' local optimum close to the global optimum.

Tao and Zhao [39] describe a partitioning algorithm based on a combination of local heuristic searches and more global random search methods called 'Stochastic Probe'. They categorise heuristic algorithms in the following way:

i. Kernighan-Lin heuristics: improving on an initial partition through repeated sequences of moves, a local, aggressive search method.

ii. Simulated annealing: see Section 2.7.2. A stochastic optimization approach which can theoretically find the global optimum but practically is too slow for most problems.

iii. Tabu search: aggressive local search algorithms which keep a history of the solution space already searched to avoid that sub-space in future moves.

iv. Genetic-based algorithms: Genetic search finds starting points for aggressive local

searching and this local search biases the choice of subsequent starting points using genetic search.

Clustering And Classification

The technique of dividing a set of data into groups is widely practised and is the subject of an entire discipline within statistics. The methods outlined above for partitioning graphs come entirely from engineering disciplines and in the light of the popularisation of other statistical methods in engineering by Taguchi and others(see Section 1.3.2) an attempt should be made to integrate the subject of clustering with methods developed in engineering. Cormack [12] provides an extensive review of the use (and misuse) of classification techniques within the scientific community.

For the case of system decomposition the requirements of a clustering package are more precise than for the case of classification in general. Here the goal is to improve the efficacy of analysis of a system through decomposition. We are thus looking for subsystems which are easy to analyse in isolation and easy to recombine to produce a model of the full system. In the limit the most desirable scenario is to be able to decompose a system into subsystems which can be analysed independently of all other subsystems with the individual results providing the analysis of the whole. However this is seldom, if ever, likely to be the case since we are, by definition, dealing with a set of connected items. As the lack of interactions between subsystems makes analysis much simpler, any clustering routine should attempt to minimize these. An advantage in using clustering techniques is that the problems of NP-completeness are avoided if we consider a clustering technique which builds clusters systematically. This makes clustering

attractive for problems such as VLSI layout [1].

A Metric For Clustering

In order to cluster a system into a set of subsystems a metric needs to be found. The metric is a measure of how similar individual components are and tells the cluster routine which components belong to which groups and how many groups there are. Without precise information on how each component interacts with every other it is difficult to cluster a system to minimize interactions. If a system is represented graphically as a set of connected components a basic relationship between components can be stated, in terms of how connected each component is to every other, using a suitable metric. Establishment of a more accurate relationship would require more information on the nature of the system components and is potentially costly to compute. Using the idea of connectivity a distance matrix associated with a system graph can be generated. This can the be used in a clustering algorithm to partition a system. Representing a system as a graph $\mathcal{G}(X;E)$ with a set of nodes $X = x_1,\ldots,x_m$ and edges $E = e_1,\ldots,e_p$ we can construct an $m \times m$ distance matrix D with elements $d(i,j)$ where $i,j = 1,\ldots,m$ represent the graph nodes. There are $2p$ entries because the matrix is symmetric about the main diagonal (as in Section 2.4.2). The matrix elements are assigned as follows:

$$d(i,j) = \begin{cases} 0 & \text{for } i = j \\ 1 & \text{for } i,j \text{ connected} \\ 9 & \text{for } i,j \text{ not connected} \end{cases} \tag{2.12}$$

Unconnected nodes are assigned a relatively high number (9 in this case) representing

their disconnection with a large distance. The initial distance matrix D can then be extended to represent higher level connections showing the shortest distance of every node from every other in the graph by recursively using the following algorithm for every higher level required:

i. Select row i for $i = 1$ to n

ii. Select element $d(i,j)$ for $j = 1$ to n

iii. For all $d(i,j) = 1$go to row j , find all elements $d(j,k) = 1$ for $k = 1$ to n ($k \neq i$).

iv. For all $d(j,k) = 1$ found in iii. if $d(i,k) > d(j,k) + 1$ then $d(i,k) = d(j,k) + 1$.

v. Repeat for all rows in matrix.

Thus one can cluster a system by grouping together highly connected components using readily available graphical information and use this concept of connectivity as a method for minimising interactions between groups, the connectivity of the system components being used as an estimate of the interaction between them. As a first approximation this estimate is valid since if there are two components of a system that are not connected then there is no interaction between them. However for a system such as an electronic circuit all components are connected to all others and interact with each other to varying degrees. The metric in this case would still minimize interactions *if* the distance between components is related to the strength of interaction. Bandler and Zhang [3] attempt to measure the interaction between system variables for problem decomposition from a system defined with parameters $\Phi = \phi_1, \ldots, \phi_n$ and outputs $Y = y_1, \ldots, y_m$ with a corresponding target response $T = t_1, \ldots, t_m$. From an initial Monte Carlo (Section 1.2.6) sensitivity analysis, construct a sensitivity matrix S where S_{ij} is the

sensitivity of variable ϕ_i to the function f_j where $f_j = y_j - t_j$ and use this to group system parameters for optimization. Further work on clustering could utilise this scheme for automatic system decomposition without referring to system topology.

Algorithms

Once a distance matrix has been generated for a given system graph an algorithm is used to cluster it. Hartigan [21] describes four joining algorithms which seek to pair up 'close' points making a single new point from them until only one point exists. This can be represented in the form of a tree showing the path from the full set of points to a single point. The four algorithms are:

i. Single Linkage: $d(ij,k) = \min d(i,k), d(j,k)$

ii. Complete Linkage: $d(ij,k) = \max d(i,k), d(j,k)$

iii. Average Linkage (unweighted): $d(ij,k) = \frac{1}{2}[d(i,k) + d(j,k)]$

iv. Weighted Average Linkage: $d(ij,k) = \frac{n_i d(i,k) + n_j d(j,k)}{n_i + n_j}$

where $d(ij,k)$ is the distance between the newly joined i,j and k , n_i is the number of original objects in cluster i.

Optimization Methods

The decomposition of a system is an optimization problem where the quality of the partition is encapsulated in an objective function. In the case of min-cut partitioning the objective function calculates the size of the cut-set of the network given a partition. As previously stated the partitioning problem is NP-complete, however optimization

methods, in particular simulated annealing, can still be used to find solutions [22].

Simulated Annealing

Simulated Annealing (SA) [23] provides a method optimization which attempts to escape from locally optimal solutions by allowing moves which are 'bad' in an attempt to find the global optimum in solution space. As the algorithm progresses this feature is gradually reduced (cooling) so that an optimal solution is found.

SA can be used in conjunction with a heuristic style approach (section 5.3) where cells are moved in an attempt to improve the partition. The result is an algorithm which can move cells which (hopefully) only temporarily worsen the quality of partition in the search for an ultimately better solution [22].

The efficacy of this method depends on the rate of 'cooling' of the algorithm but in general it is time consuming and cannot find the global solution in a practical time frame [9].

2.7.3 Tearing and Diakoptics

The idea of dividing, or tearing, a network into smaller parts to ease numerical calculation was explored by Kron [25] in a series of articles published in the 'Electrical Journal' collectively known as *Diakoptics* (literally meaning 'system tearing'). Other work related to the exploitation of sparse matrices [20, 32] also promotes the idea of decomposing systems (particularly electronic circuits) through an exactly analogous decomposition of the incidence matrix representing the graph of the system or network. Decomposition through tearing follows the method of Diakoptics in defining sub-systems

with an interconnecting network of components. By formulating the incidence matrix of a circuit and translating it to BBD form one is dividing the circuit network in the same way as for Diakoptic analysis. The purpose of Diakoptics is to formulate the equations of the system under investigation in an efficient, piecewise manner.

Considering the system as being formulated in terms of an electrical network, the system equations take the form:

$$I = Y.E \tag{2.13}$$

where I is the current vector, Y the nodal admittance matrix and E the voltage vector (see Chapter 2 for formulation of Kirchoff equations in matrix form). The problem is then, given Y and I, to find E. This involves inversion of Y and can be costly in computer time for large matrices. The effect of decomposing the network is to produce several smaller admittance matrices, rather than one big one, making the inversion process easier. An outline of the method follows:

i. Tear the system into n sub-networks.

ii. Formulate the system equations for each sub-network obtaining $Y_1 \ldots Y_n$.

iii. Solve the equations obtaining the inverses of $Y_1 \ldots Y_n$, call them $Z_1 \ldots Z_n$.

iv. Establish and solve the $(n+1)$th network, the connecting network, obtaining the connecting matrix C and the inverse matrix Z_{n+1}.

v. With these inverses $Z_1 \ldots Z_{n+1}$ computed and the connection matrix C the system is considered solved.

The BBD form of the incidence matrix generated by the algorithm in Section 2.4.2

represents this decomposition technique exactly. The blocks of the incidence matrix, when viewed on the system graph represent the groups of (i.) and the connection matrix in (iii.) is given by the border of the incidence matrix. The strategy is to use the BBD decomposition to form sub-systems and to represent interactions between them through the connecting network.

Diakoptics begins by partitioning a network representing the system of interest. Research into this area focuses on how to tear the network to maximize efficiency and has led to the use of heuristics [35] and optimization methods [22] in the partitioning (tearing) procedure. Emphasis has been placed on the numerical techniques used to solve the equations and has led to the application of sparse matrix techniques [20] to improve efficiency of analysis.

Diakoptics has also been employed in circuit optimization to improve the efficiency of obtaining first and second order sensitivity information for non-linear networks [40].

2.7.4 Optimization of decomposed systems

Direct Decomposition Methods (DDM) and Hierarchical Decomposition Methods (HDM) manipulate the system equations into blocks of equations related algebraically or by an overall control block. An example of this is Diakoptics[25] discussed in Section 2.7.3.

Direct decomposition methods

Direct decomposition is where a graph $\mathcal{G}(X;E)$ with a set of nodes $X = x_1,\ldots,x_m$ and edges $E = e_1,\ldots,e_p$ representing a system is taken and split into subgraphs. This can occur in three ways [37]:

i. Node Decomposition - where a graph is split through the nodes forming n separate subsets of edges $E_1, \ldots, E_n$ representing subgraphs. Nodes shared between subgraphs are called block nodes represented by the set X_b.

ii. Edge Decomposition - where a graph is split through the edges forming n separate subsets of nodes $X_1, \ldots, X_n$ representing subgraphs. Edges shared between subgraphs are called cutting edges represented by the set E_c.

iii. Hybrid Decomposition - a mixture of the two categories above.

After division into blocks another graph, the decomposition substitute graph, containing the sets X_b and E_c is formed to preserve information on how the partitions are connected to reconstruct the full system. As system size increases there is a conflict between subgraph size and substitute graph complexity. In the limit direct decomposition methods do not work well enough to efficiently partition systems as their size increases [37]. To deal with these more complex systems Hierarchical Decomposition Methods are used.

Hierarchical decomposition methods

Hierarchical Decomposition is the multiple decomposition of a decomposed network. It is used where direct decomposition methods are unable to tackle problems efficiently and so tends to be used for large or complex systems.

Hierarchical Decomposition involves applying simple decomposition recursively to subgraphs to keep the decomposition substitute graphs simple while allowing the system subgraphs to be reduced to a manageable size. Once a hierarchical structure of subgraphs and decomposition substitute graphs is obtained it is analysed to produce a

description of the system. Analysis can be achieved by either working from the lowest subsystem up the tree hierarchy to the top, 'bottom up', or from the top of the structure downwards, 'top down'. Starzyk [37] compares these two approaches and provides an algorithm for the 'bottom up' method. Note that, for analysis of the whole system, the subsystems are recombined for solution.

To coordinate the solution of the subproblems there are two basic approaches:

i. Goal Co-ordination - where the objectives of each subproblem are controlled.

ii. Model Co-ordination - where the interactions between subsystems are identified and assigned co-ordination variables handled by the controller of the interacting subsystems.

Overall system stability is considered in [9] for a hierarchically decomposed system. For a system sub-divided into 'strongly connected subsystems' by a partitioning algorithm it is shown that the overall system is stable when the individual subsystems and the interconnection subsystems are stable.

2.8 Conclusions

The development of simulation techniques has been covered and the use of circuit simulators for RED discussed. The area is characterized by the application of mathematical techniques both for the development of new simulation methodologies and the enhancement of existing ones with the common goal of simulating circuits more efficiently allowing the size of solvable problems to increase. The specific requirements of RED allow further savings in the cost of simulation when used to conduct experiments

and this points the way to the development of simulation software in tandem with design systems to achieve an optimally efficient package for the design of robust systems. Several approaches to the problem of reducing large, complex systems into subsystems for analysis have been outlined. The separate areas of decomposition and analysis and their inter-relationship have been identified. The quality of any decomposition depends on the type of analysis to be employed afterwards and the ability of the algorithm to find a good local optimum close to the globally optimal solution. Heuristic algorithms, clustering and general optimization methods can all be used to partition systems with different methods suited to different applications all, however, produce locally optimal solutions. The production of a global optimum requires an impractical amount of time but is nevertheless possible with optimization methods such as simulated annealing. Heuristic solutions are more practical with respect to time.

Once partitioned a system can be analysed according to the methods outlined, the sub-systems being solved simultaneously to converge to a solution taking interactions between partitions into account. The use of decomposition within Robust Engineering Design requires the sub-systems to be analysed independently for any gain in efficiency. The solution of sub-systems independently fails to deal with interactions, the assumption being that the main effects of parameters in the sub-systems are more influential on system response than interactions between parameters of different partitions. Thus the quality of partitioning of a system plays a direct role in the accuracy of analysis in this case.

References

[1] S B Akers. Clustering techniques for VLSI. *IEEE Trans. Computers*, 33(5):472–476, 1982.

[2] J W Bandler, R M Biernacki, Qian Cai, S H Chen, Shen Ye, and Qi-Jun Zhang. Integrated physics-oriented statistical modelling, simulation and optimization. *IEEE Trans. Microwave Theory and Techniques*, 40(7):1374–1399, 1992.

[3] John W Bandler and Qi-Jun Zhang. An automatic decomposition approach to optimization of large microwave systems. *IEEE Trans. Microwave Theory & Tech.*, 35:1231–1239, Dec 1987.

[4] R A Bates, R J Buck, E Riccomagno, and H P Wynn. Experimental design and observation for large systems. *Journal of the Royal Statistical Society B*, to appear.

[5] Maria C. Bernardo, Robert Buck, Lishin Liu, William A Nazaret, Jerome Sacks, and William J Welch. Integrated circuit design optimization using a sequential strategy. *IEEE Trans. Computer Aided Design.*, CAD-11:361–372, 1992.

[6] R K Brayton and R Spence. *Sensitivity and optimization*, chapter 6, page 134. Elsevier, Amsterdam, 1980.

[7] R J Buck. Robust Engineering Design Users Guide. Technical report, City University Engineering Design And Quality Centre, Northampton Square, London, 1993.

[8] R J Buck. *The design and analysis of computer experiments.* PhD thesis, City University, London, UK, 1994.

[9] Frank M Callier, Wan S Chan, and Charles S Desoer. Input-output stability theory of interconnected syatems using decomposition techniques. *IEEE Trans. Circuits & Systems*, CAS-23(12):714–729, Dec 1976.

[10] Chung-Kuan Cheng. The optimal partitioning of networks. *NETWORKS*, 22:297–315, 1992.

[11] Leon O. Chua and Pen-Min Lin. *Computer Aided Analysis of Electronic Circuits : algorithms and computational techniques.* Prentice Hall Inc., Englewood Cliffs, New Jersey, 1975.

[12] R M Cormack. A review of classification. *Journal Royal Stat. Soc. B*, pages 321–353, Mar 1971.

[13] Madhav P Desai and Ibrahim N Hajj. On the convergence of block relaxation methods for circuit simulation. *IEEE Trans. Circuits & Systs.*, 36:948–958, July 1989.

[14] H Eltahawy, S Garciasabiro, D Rodriguez, and J J Mayol. Towards an analog hardware description language - based on VHDL. In *Proceedings of the 1994 Western Multiconference*, pages 48–53. International Conference on Simulation and Hardware Description Languages (ICSHDL), 1994.

[15] C M Fiduccia and R M Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181, 1982.

[16] M R Garey and D S Johnson. *Computers and Interactability : a guide to the theory of NP-completeness.* W H Freeman and Co., 1979.

[17] Georges E Gielen and Herman C C Walscharts. Isacc : A symbolic simulator for analog integrated circuits. *IEEE Journal of Solid State Circuits*, 24:1587–1597, Dec 1989.

[18] Georges E Gielen, Herman C C Walscharts, and Willy C Sansen. Analog circuit design optimization based on symbolic simulation and simulated annealing. *IEEE Journal of Solid State Circuits*, 25:707–713, June 1990.

[19] Gary D Hachtel and Alberto L Sangiovanni-Vincentelli. A survey of third-generation simulation techniques. *Proceedings of the IEEE*, 69:1264–1280, Oct 1981.

[20] Ibrahim N Hajj. Sparsity considerations in network solution by tearing. *IEEE Trans. Circuits & Systs.*, CAS-27:357–366, May 1980.

[21] J A Hartigan. Distribution problems in clustering. In J Van Ryzin, editor, *Classification and Clustering*, pages 45–71. Academic Press, Inc., London, 1977.

[22] M R Irving and M J H Stirling. Optimal network tearing using simulated annealing. *IEE Proceedings Pt-C*, 137:69–72, Jan 1990.

[23] S Kirkpatrick, C D Gelatt, and M P Vecchi. Optimization by simulated annealing. *Science*, 220:671–679, May 1983.

[24] Balakrishnan Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. Computers*, C-33:438–446, May 1984.

[25] G Kron. *Diakoptics: The Piecewise Solution of Large Scale Systems.* MacDonald, London, 1963.

[26] Kenneth S Kundert and Alberto Sangiovanni-Vincentelli. Simulation of nonlinear circuits in the frequency domain. *IEEE Trans. Computer Aided Design*, CAD-5:521–535, Oct 1986.

[27] P M Lin. A survey of applications of symbolic network functions. *IEEE Trans. Circuit Theory*, CT-20:732–737, Nov 1973.

[28] L W Nagel. *SPICE 2. A computer program to simulate semiconductor circuits.* ERL Memo ERL-M520. Univ. California, Berkley, 1975.

[29] Arthur Richard Newton and Alberto L Sangiovanni-Vincentelli. Relaxation based electrical simulation. *IEEE Trans. Electron Devices*, ED-30:1184–1207, Sep 1983.

[30] K G Nichols, T J Kazmierski, M Zwolinski, and A D Brown. Overview of SPICE-like circuit simulation algorithms. *IEE Proc.-Circuits Devices Syst.*, 141(4):242–250, 1994.

[31] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* CBMS-NFS, SIAM, Philadelphia, 1992.

[32] E C Ogbuobiri, W F Tinney, and J W Walker. Sparsity-directed decomposition for gaussian elimination on matrices. *IEEE Trans. Power Apparatus & Systems.*, PAS-89:141–150, Jan 1970.

[33] Donald O Pederson. A historical review of circuit simulation. *IEEE Trans. Circuits & Systs.*, CAS-31:103–111, Jan 1984.

[34] Jerome Sacks, William J Welch, Toby J Mitchell, and Henry P Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–435, Nov 1989.

[35] Alberto Sangiovanni-Vincentelli, Li-Kua Chen, and Leon O Chua. An efficient heuristic cluster algorithm for tearing large-scale networks. *IEEE Trans. Circuits & Systs.*, CAS-24:709–717, Dec 1977.

[36] Kishore Singhal and Jiri Vlach. Symbolic analysis of analog and digital circuits. *IEEE Trans. Circuits & Systs.*, CAS-24:598–609, Nov 1977.

[37] Janusz A Starzyk and A Konczykowska. Flowgraph analysis of large electronic networks. *IEEE Trans. Circuits & Systs.*, CAS-33:302–315, Mar 1986.

[38] Fang K T and Wang Y. *Number-theoretic Methods in Statistics.* Chapman & Hall, London, 1994.

[39] L Tao and Y C Zhao. Effective heuristic algorithms for VLSI circuit partition. *IEE Proceedings-G*, 140(2):127–134, April 1993.

[40] Paul K U Wang, Chin Fu Chen, and Yuang-Sheng Kao. Sensitivity calculation and network optimization through decomposition. *Proc. IEEE Int. Symp. Circuits & Systems*, 3:1034–7, 1983.

[41] W. J. Welch, R. J. Buck, J. Sacks, H. P. Wynn, T. J. Mitchell, and M. D. Morris. Screening, predicting, and computer experiments. *Technometrics*, 34(1):15–25, 1992.

[42] H. P. Wynn and A. A. Zhigljavsky. *Fundamentals of Search.* Springer-Verlag, New York, to appear.

[43] Jih-Shyr Yih and Pinaki Mazumder. A neural network design for circuit partitioning. *IEEE Trans. Computer Aided Design.*, CAD-9:1265–1271, Dec 1990.

[44] A I Zečevič and D D Šiljak. Balanced decompositions of sparse systems for multilevel parallel processing. *IEEE Trans. Circuits & Systems-I: Fundamental Theory & Applications.*, 41(3):220–233, March 1994.

[45] Anatoly A. Zhigljavsky. *Theory of Global Random Search*, chapter 4. Kluwer Academic Publishers, 1991.

Chapter 3

Robust circuit design I:

A commercial environment

3.1 Introduction

The use of computers is widespread in engineering design with a multitude of CAE/CAD tools available. There are several commercially available software packages providing tools for the design and analysis of both analogue and digital electronics products. Typically analogue packages include features such as schematic capture, simulation and auto-routing for the input, testing and layout of a design. A designer will use the circuit simulator to check that the design performs as intended and there are also tools, such as Monte Carlo analysis, which give information about the sensitivity of the design to manufacturing tolerances. In this chapter a system for using Robust Engineering Design with an analogue circuit simulator is described which provides the circuit designer with a powerful Robust Circuit Design (RCD) tool for circuit optimization. This is

demonstrated with an example.

The RCD package described in this chapter is the result of a collaborative project between Mentor Graphics UK Ltd. and the Engineering Design and Quality Centre at City University which acted as an initial project into the application of RED to the commercial environment. The well-proven modules for experimental design and model-building are intended as as introduction to RED, the more sophisticated techniques being applied throughout the rest of the thesis. The main contribution of this Chapter thus lies in providing a unified framework for the execution and analysis of RED experiments.

3.2 Overview

3.2.1 The simulator

The Mentor Graphics software is comprised of several tools for the design, analysis and manufacture of circuits under a common framework. The module of interest here is the analogue circuit simulator AccuSim, based on the well known SPICE circuit analysis package developed by Nagel [8]. In order to facilitate communication between different modules and the development of functions to control the software and perform general tasks the framework provides a formal language called Ample in which all high-level functions are written. The user is also free to develop specialist functions within the framework in Ample to control the software.

The facilities of AccuSim, the analogue circuit simulator, include all the main analysis options of SPICE (DCOP, DC, AC and transient analyses) integrating this with schematic capture of circuit diagrams, a library of equivalent circuit models for

non-linear components and other features such as Monte Carlo analysis in an integrated windows environment.

3.2.2 Robust design modules

With the facilities of Ample in mind a suite of functions were developed in C to design experiments for RED and analyse results from them. Because of the high-level nature of Ample a more efficient computational solution for the RED calculations is to use C. This also has the advantage of a wider range of mathematical functions and debugging facilities for development. These functions are integrated into the Ample language and provide the technical content of the RED software. The three modules are:

i. 3^k fractional factorial experimental design plan generator.

ii. Latin Hypercube Sampling experimental design plan generator.

iii. Analysis package to provide factor plots and regression models.

These modules will be fully described in section 3.3.

3.2.3 Interface

The AMPLE language is similar to the computer language C++ in structure and provides access to the commercial simulation software. As well as providing specialist commands to control this software, AMPLE contains all the basic commands associated with a language such as C. An important additional feature is the ability to 'build-in' C functions into the Ample code, this allows the RED software to be developed in C and then linked to the interface. Paramount in the conception of the package was the idea of creating a modular framework so that different RED functions could be used with the

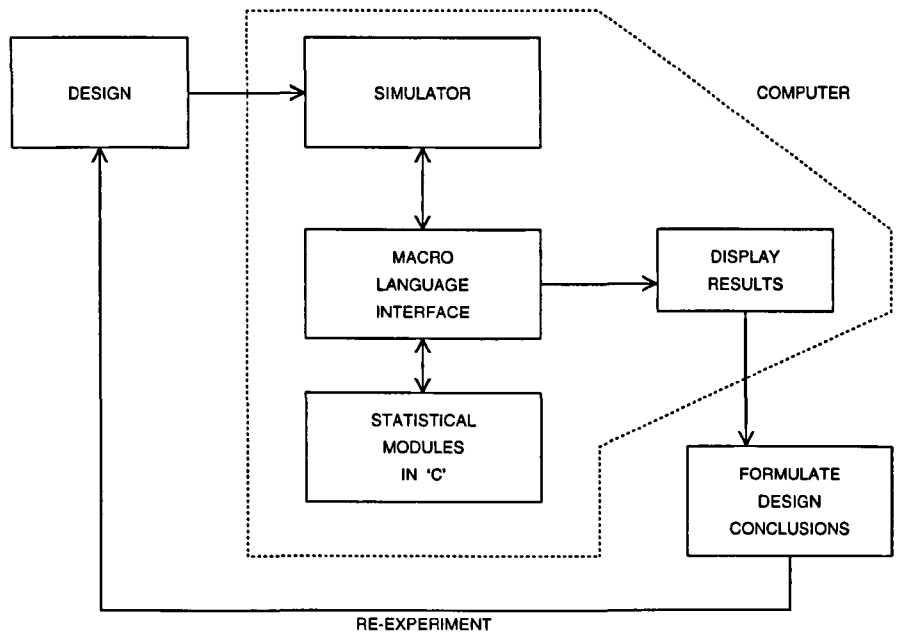


Figure 3.1: Schematic of RCD module

commercial software once the interface was complete. This allows the latest developments in RED to be used in an efficient way by simply changing the C modules developed. The interface therefore performs the task of controlling both the simulator and RED software and translating all the data required between them. Figure 3.1 shows a schematic of the project structure.

3.2.4 Output

The results of the RED experiment will be in two forms. First a set of plots will be displayed describing the effect of component variation on the chosen output. Second a

regression model will be fitted to the data for optimization.

3.2.5 Optimization

A numerical optimizer is not included in the RCD package but, given a suitable optimizer, it is possible to use the models built during the analysis phase to optimize the design. Chapter 4 describes a novel method of tolerance design (tolerance design is reviewed in Section 1.2.5) developed as a follow-on package to the RCD module.

3.2.6 RED process

The overall robust design process described in Section 1.3 is adopted as:

i. Given a circuit with parameters at an initial nominal setting and tolerance, use the simulator to obtain the required output $Y = f(X)$ for inputs $X = x_1, \ldots, x_n$ set according to an experimental design plan.

ii. Fit a regression model $\hat{Y} = \hat{f}(X)$, this is the *emulator* of the simulator (as defined in Section 1.2.3).

iii. Use the emulator to find inputs X_t which bring $\hat{Y}$ to within some target value Y_t. A numerical optimizer is used in the last step to search X-space for a solution (see Chapter 4).

iv. Confirm the solution with the simulator. If greater accuracy is required reduce the input space (tolerances) and repeat the above steps at the optimized nominal settings.

3.3 The RCD modules

This section describes the technical content of the package and concludes with a brief users guide. Using the software can be summarised in the following steps

i. select circuit parameters for inclusion in the experiment

ii. create a design plan

iii. execute the experiment and collect relevant results

iv. build an emulator of the circuit simulator

v. display factor plots

3.3.1 Circuit parameters

The simulator contains two libraries of components. The generic library contains linear components (resistors, capacitors etc.) and equivalent-circuit models of non-linear devices built from combinations of components (transistor models etc.). The second library contains proprietary models of non-linear devices representing commercially available components.

Changing the value of linear components in the generic library is directly related to the component values for a real circuit design. This is not the case for devices from the model library where parameters of device models do not represent physical characteristics of the devices they represent. Because the models are often not available for inspection, it is difficult to attach any meaning to changing model parameter values as part of a Robust Design experiment. Varying model parameters does not necessarily mimic the manufacturing variation of device parameters. To counter this the simulator

library provides variations on particular device models but this can only deal with particular device characteristics and this may only be useful in certain situations. The RCD package allows the variation of device parameters as part of an RED experiment, providing some insight into how sensitive the overall design is to changes in the device *model*, however it should be stressed that they do not represent manufacturing parameters or tolerances. Relating device models to manufacturing data and process models is a problem for the whole electronics design community and a suggested area for further research.

3.3.2 Experimental design

The RCD package provides a choice of two types of design plan. Written in 'C' they take information from the simulator and return an appropriate design plan for the experiment.

3^k designs

The 3^k designs used are three-level Plackett-Burman designs which are specially designed orthogonal arrays. For a parameter, or factor, p taking values $x \pm t\%$ the three levels represent $x - t\%$, x, $x + t\%$. A design plan where each factor is tested at each of these three levels can produce a prohibitively large design plan even for small problems. One way to reduce design plan size is to use orthogonal arrays where some combinations of factor levels will be missed out leading to a reduced design. 3^k designs are useful for estimating the average effect of each factor on the output, called the *main effects*, when one does not expect interactions between factors. An example 3^k Plackett Burman design for four factors x_1 to x_4 at levels $-1, 0, +1$ is displayed in Table 3.1. The code for choosing the designs is written in 'C' while the designs themselves are stored in an

Run	x_1	x_2	x_3	x_4
1	-1	-1	-1	-1
2	-1	0	0	0
3	-1	1	1	1
4	0	-1	0	1
5	0	0	1	-1
6	0	1	-1	0
7	1	-1	1	0
8	1	0	-1	1
9	1	1	0	-1

Table 3.1: 3^k design

ASCII file. The interface language Ample provides access to the simulator.

Latin Hypercube Sampling (LHS) designs

A description of LHS designs is given in Section 2.3.1, here we describe the implementation of an LHS generator within the RCD framework. The LHS design is created from a combination of randomised vectors of factor values. The size of the design can be changed in the program but is initialised at the suggested value of $2d + 10$ for d factors. This allows estimation of main effects and a small number of interactions between factors.

The factor values are expressed as a nominal value with a relative tolerance attached (see Section 4.3 for a discussion of tolerances). For a factor p taking values $x \pm t\%$ the vector is created by first forming a vector of length $2d + 10$, filling it with evenly spaced numbers in the range $x - t\%$ to $x + t\%$ and randomising it.

An example LHS design for four factors x_1 to x_4 in the space $[-1, 1]^4$ is shown in Table 3.2

The code for creating the LHS designs is written in 'C' and accessed by the interface

Run	x_1	x_2	x_3	x_4
1	-0.278	0.056	-0.278	-0.056
2	0.500	-0.389	0.611	0.167
3	-0.167	-0.167	0.389	0.944
4	0.611	-0.944	-0.167	-0.389
5	0.278	-0.500	-0.833	0.500
6	-0.500	-0.722	-0.056	0.722
7	-0.389	0.167	-0.500	-0.833
8	-0.833	-0.056	-0.611	0.389
9	-0.944	0.833	0.278	0.611
10	-0.722	-0.833	-0.944	0.056
11	0.389	0.389	0.056	0.833
12	0.722	0.500	0.944	-0.278
13	-0.611	0.722	0.500	-0.611
14	0.167	0.944	0.722	-0.722
15	0.833	-0.278	-0.389	-0.500
16	-0.056	-0.611	-0.722	-0.944
17	0.056	0.278	0.833	0.278
18	0.944	0.611	0.167	-0.167

Table 3.2: LHS design

language Ample.

3.3.3 Circuit outputs

The simulator software provides functions for measuring several standard circuit responses. These differ depending on the type of simulator analysis chosen, the available responses include

AC analysis Bandpass highpass, lowpass, peak frequency, peak magnitude, stopband, trough frequency, trough magnitude, maximum, minimum, signal to noise ratio, point voltage.

Transient analysis Baseline, crosspoint, delay time, distal, duty, fall time, frequency, mesial, overshoot, period, proximal, rise time, settle time, slewrate, topline, undershoot, maximum, minimum.

Several responses can be chosen for a single experiment. The values of each response are calculated and stored for every trial.

3.3.4 The emulator

A model is built for each response chosen in the experiment. The model is an emulator of the simulator for that particular response over the ranges of input values chosen. It is much less expensive to evaluate than the simulator and can be used effectively in the objective function of an optimization routine (see section 3.3.5). The emulator is used to create factor plots describing the effect of component variation on circuit response, see Figure 3.4 for an example.

3.3.5 Optimization

The emulator can be used to build an objective function for inclusion in a global optimizer, see Section 2.5. The case study (Section 3.4) shows how the factor plots generated by the RCD package can be used as an initial guide to improving the design while Chapter 4 provides a framework for global design optimization.

3.3.6 Using the RCD module

The RCD package is started by selecting the 'RCD' (Robust Circuit Design) option from within the AccuSim simulator. Once selected the software prompts the user for information about the circuit design needed before an experiment can begin. This is collected and then used to execute the experiment and analyse the results in the following order.

Initialization of the simulator

The user is prompted to apply forces to the circuit where necessary appropriate for the analysis to be performed and then choose the type of simulation from the standard SPICE-style list of DCOP, DC, AC or transient analyses. The range of time or frequency values is also required along with the number of points per interval to simulate at. This is important in accounting for the accuracy of simulation results.

Input parameters

Circuit components (parameters) that are to be included in the experiment are chosen here. The software accepts parameters from both linear and non-linear components which are selected by highlighting them on the circuit schematic. In the case of non-linear components, because they are represented by equivalent circuit models, the user is asked how many parameters within the associated model-file they wish to vary, the nominal value is then required for each parameter. The % tolerance values are then required for each component with a default option of 10%, this will be the amount that the nominal value of each parameter will vary by during the course of the experiment.

Design plan

The design plan gives, for each trial of the experiment, the values for each input parameter. There are two types of design plan to choose from.

i. 3^k *designs.* 3^k designs are chosen from a lookup table using a 'built-in' C function "pick_design". The lookup table contains a set of design plans which can handle experiments with up to 40 input parameters. 3^k designs are used in cases where a

basic estimation of main-effects is needed for sensitivity analysis using as few trials as possible.

ii. *Latin hypercube sampling designs.* Latin hypercube sampling (LHS) designs are generated from the 'built-in' C function "make_design". LHS design plans can be generated for experiments with any number of input parameters and, because of their good space-filling properties and ability to estimate more than just main-effects (see Section 1.3.2), are the preferred choice of plan for experimentation.

Outputs

After each trial of the experiment the RED software stores the circuit responses of interest to the designer. These responses (outputs) are selected from a standard set supplied by the simulator software.

Analysis

On completion of the experiment another 'built-in' C function "analysis" is used to build a polynomial regression model emulator of the circuit. This provides a less adaptive emulator than the DACE model emulator described in Section 2.3.2 but is more attractive for this application because of its easy implementation. Model building is achieved through the use of a mixture of forward and backward variable selection methods. In the case of 3^k designs, due to the orthogonal nature of the design (see Chapter 1) a model of main and second order effects is built (no interactions) and for LHS designs a full quadratic model is built. The model is then used to produce factor

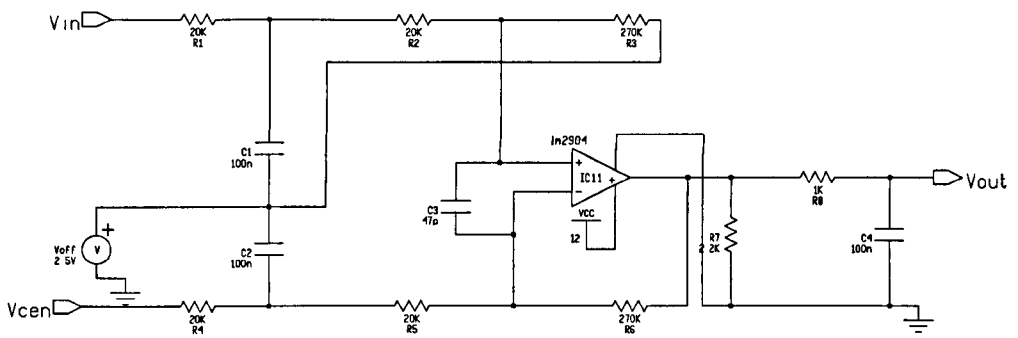


Figure 3.2: Voltage amplifier circuit

plots showing the influence each input parameter has on each output chosen. This provides a quick visual indicator of the sensitivity of the design to parameter variation and how to change the values of sensitive parameters to reduce this.

More comprehensive optimization can be achieved by using the regression model combined with a numerical global optimizer. Because the regression model or 'emulator' is easy to compute this makes for more efficient circuit optimization than using the simulator directly.

3.4 A case study

3.4.1 Introduction

To illustrate the RCD procedure the circuit of Figure 3.2 was input to the simulator for analysis. The circuit is a voltage amplifier designed as part of an electric wheelchair controller unit. By measuring the voltage across the tracks of a printed circuit board and inputting the amplified voltage to a microprocessor the controller estimates the supply

Monte Carlo	Initial design	Tolerances adjusted
Mean	3.863	3.862
Variance	2.822e-4	2.924e-4

Table 3.3: Summary of Monte Carlo confirmatory experiments

current to the wheelchair motor. Being part of an existing design the circuit components already have a set of nominal values and as the whole wheelchair design is in a safety-critical environment all component tolerances are set to $\pm 1\%$.

The aim of the RCD study is twofold:

Nominal design To check the operation of the circuit under manufacturing conditions and see if this can be improved by changing the nominal values of the design.

Tolerancing To establish which are the most important components so that tolerances can be assigned according to the sensitivity of the circuit response to each component.

For this part of the case study the results will be used to identify which components affect response the most and to adjust their tolerances accordingly. The issue of changing the nominal values of the design is discussed in Chapter 4 where the emulator built as part of the RCD process is used for design optimization.

3.4.2 Experimentation

As a first step a 200 run DC Monte Carlo analysis is carried out on the original nominal design with 12 component tolerances set to a Gaussian distribution of $\pm 1\%$. The histogram of Figure 3.3 shows the performance of the circuit, the mean output voltage, $\mu_v = 3.863$ with associated variance estimate $\hat{\sigma}_v^2 = 2.822 \times 10^{-4}$.

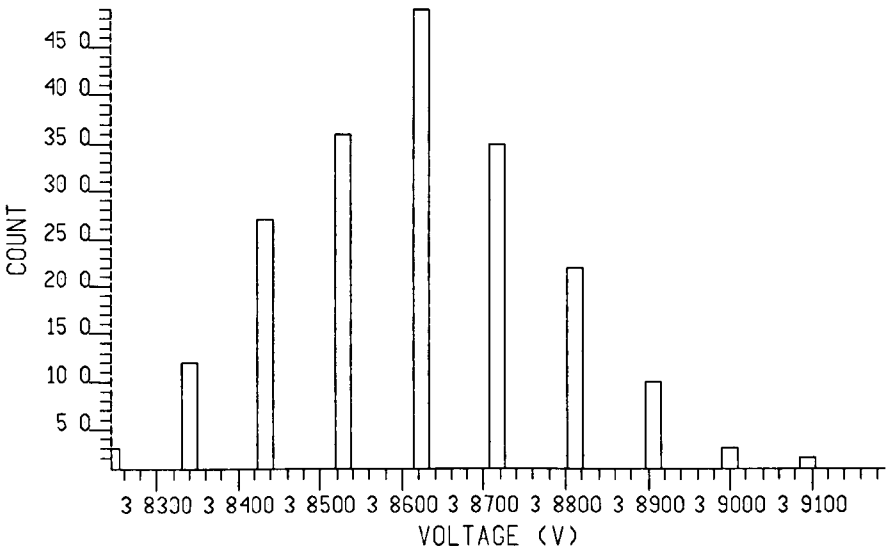


Figure 3.3: Histogram of voltage output from the Mentor Graphics software

Component	Initial design		Toleranced design	
name	Nom.	Tol.	Nom.	Tol.
R1	270K	±1%	270K	±1%
R2	20K	±1%	20K	±1%
R3	20K	±1%	20K	±1%
R4	270K	±1%	270K	±1%
R5	20K	±1%	20K	±1%
R6	20K	±1%	20K	±1%
R7	2.2K	±1%	2.2K	±20%
R8	1K	±1%	1K	±20%
C1	100n	±1%	100n	±20%
C2	100n	±1%	100n	±20%
C3	47p	±1%	47p	±20%
C4	100n	±1%	100n	±20%

Table 3.4: Summary of tolerancing process

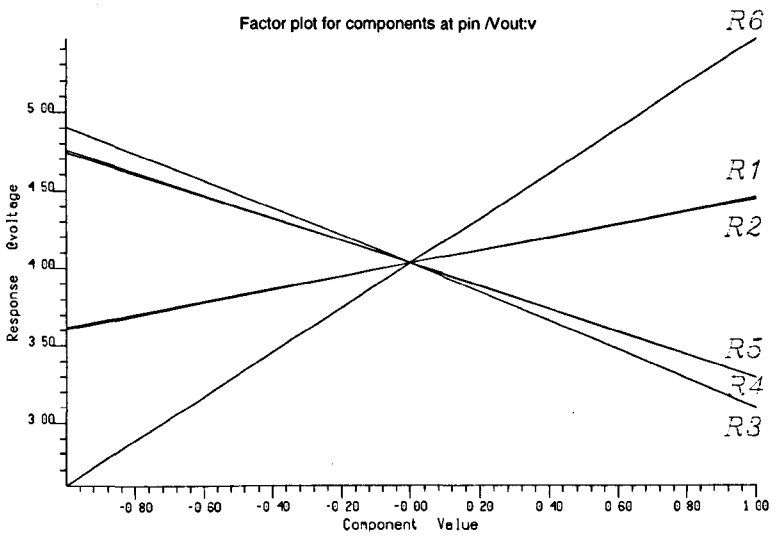


Figure 3.4: Factor plots for regression model

Next a Robust Design study was carried out using the simulator combined with the RCD software module. The 12 components were selected at the given nominal values with tolerances of $\pm 40\%$. Using a Latin Hypercube design with 50 runs the RCD experiment produced the following regression model:

$$\begin{aligned} Y \;=\; & 4.03 \;+\; 2.87 \times R215 \;-\; 1.81 \times R209 \;-\; 1.44 \times R213 \\ & -\; 1.46 \times R214 \;+\; 0.85 \times R210 \;+\; 0.82 \times R211 \\ & -\; 1.48 \times R215 \times R209 \;+\; 0.92 \times R213 \times R214 \end{aligned} \tag{3.1}$$

Where Y represents the output voltage of the circuit at Vout (Figure 3.2), all factor values being scaled to the range $[-0.5, 0.5]$. The model has been truncated to the most important factors and their interactions and accounts for 97.4% of the variation. The full regression model was used to construct the factor plots in Figure 3.4 showing the

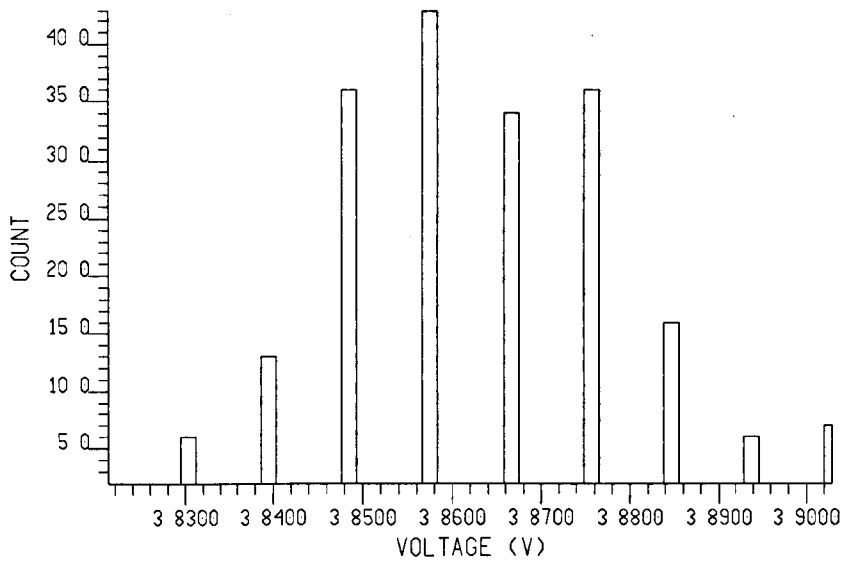


Figure 3.5: Monte Carlo histogram of results for toleranced design

importance of the six resistors. Following [1] the components are assigned tolerances which reflect their importance to the response. The tolerance of each component was adjusted according to the regression results with the six resistors in the factor plot receiving a tolerance of $\pm 1\%$ and the rest $\pm 20\%$. Table 3.3 shows the effect of adjusting the tolerances with a second 200 run Monte Carlo experiment the results of which are shown in Figure 3.5. Relaxing the tolerances of the six factors which do not affect the output response reduces the cost of manufacturing the circuit for a 3.6% increase in the variability of the response.

3.5 Discussion

The software described allows the designer to plan, execute and analyse results from an RED experiment. The separation of statistical modules, written in C, from the software

controlling the ECAD tools means that existing modules can be updated with more sophisticated software for modelling and optimization as it becomes available. The developed system eases the task of designing, executing and analysing RED experiments by providing a unified framework for circuit design. Production of Factor plots allows a quick assessment of the design acting as a guide to the first step in tolerancing the design and highlighting any possible problems with design robustness. The linearity of the factor plots for the case study show the linear effects of the component parameters on circuit response. For the case study described the factor plots were used to identify important components and allowed the tolerances of unimportant components to be relaxed from $\pm 1\%$ to $\pm 20\%$ with only a 3.6% reduction in variance for a 200 run Monte Carlo confirmation experiment on the simulator.

3.6 RED within a CAD framework

The evolution of CAD tools has benefited by the integration of different techniques through use of a unifying framework. This drive has led to the development of several systems to aid designers which share some of the features of the RCD package described in this chapter. CAD frameworks are reviewed in [4] where the development of different types of interface for handling engineering information is considered.

The combination of design optimization techniques in a single package removes a lot of the difficulty in performing experiments and numerical optimization for RED simply by unifying data handling. Packages have been developed [13, 7] which include features such as:

i. group search - locating important factors [10].

ii. design of experiments - LHS, Plackett-Burman [6], Box-Behnken [3].

iii. model building - regression, stochastic processes [11]

iv. optimization - simulated annealing [5].

The problem of device modelling referred to in Chapter 2 and in this Chapter is being tackled with systems to ease the use of device simulators [12] and frameworks for moving from process simulators such as FABRICS [9] to SPICE [8] easily [14]. The problem of integrating Physics-based device models (as opposed to the equivalent circuit models in SPICE) with circuit simulation is referred to in [2].

References

[1] K J Antreich, P Leibner, and F Pőrnbacher. Nominal design of integrated circuits on circuit level by an interactive improvement method. *IEEE Trans. Circuits & Systs.*, 35:1501–1511, Dec 1988.

[2] J W Bandler, R M Biernacki, Qian Cai, S H Chen, Shen Ye, and Qi-Jun Zhang. Integrated physics-oriented statistical modelling, simulation and optimization. *IEEE Trans. Microwave Theory and Techniques*, 40(7):1374–1399, 1992.

[3] G E P Box and D W Behnken. Some new three level designs for the study of quantitative variables. *Technometrics*, 2:455–475, 1960.

[4] D S Harrison, A R Newton, D L Spickelmier, and T J Barnes. Electronic CAD frameworks. *IEEE Proceedings.*, 78(2):393–417, 1990.

[5] S Kirkpatrick, C D Gelatt, and M P Vecchi. Optimization by simulated annealing. *Science*, 220:671–679, May 1983.

[6] Dennis K J Lin and Norman R Draper. Projection properties of plackett and burman designs. *Technometrics*, 34:423–428, 1992.

[7] G J Meidt and K W Bauer Jr. Pcrsm : A decision support system for simulation metamodel construction. *Simulation*, 59(3):183–191, 1992.

[8] L W Nagel. *SPICE 2. A computer program to simulate semiconductor circuits.* ERL Memo ERL-M520. Univ. California, Berkley, 1975.

[9] S R Nassif, A J Strojwas, and S W Director. FABRICS II : a statistically based IC fabrication process simulator. *IEEE Trans. Computer-Aided Design*, 3, 1984.

[10] J H O'Geran. *Group testing and search.* PhD thesis, City University, London, UK, 1994.

[11] Jerome Sacks, William J Welch, Toby J Mitchell, and Henry P Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–435, Nov 1989.

[12] Mark R Simpson. Pride : An integrated design environment for semiconductor device simulation. *IEEE Trans. Computer Aided Design*, 10:1163–1174, Sep 1991.

[13] Kishore Singhal, Colin C McAndrew, Sani R Nassif, and V Visvanathan. The CENTER design optimization system. *AT&T Technical Journal*, pages 77–91, May 1989.

[14] James P Spoto, W Terry Coston, and C Paul Hernandez. Statistical integrated circuit design and characterization. *IEEE Trans. Computer Aided Design*, CAD-5:90–103, Jan 1986.

Chapter 4

Design optimization

4.1 Introduction

In this Chapter a novel approach to design optimization is described and demonstrated by continuing the case study of Chapter 3. The optimization process is directly related to quality and robustness as defined in Chapter 1. Emulator models of systems are formed as part of the RED process discussed in the previous Chapter. These models are used in a global optimization strategy to improve design quality.

We shall favour this approach from the following rationale : the emulators run hundreds or thousands of times faster than many simulators and are therefore useful for performing fast, approximate optimization and sensitivity analysis.

We concentrate on using the emulator to carry out robust optimization along the lines of the recent work in Robust Engineering Design (RED) reviewed in Section 1.3.

4.2 Performance region methods

Sensitivity analysis and optimization have been conducted with computer simulators using a variety of methods. Circuit optimization is reviewed in Section 1.2. For clarity the various approaches are summarised in this section as 'performance region' methods as an introduction to the optimization method developed in this chapter. From Section 1.1.1 for a given system the relation of input $X = (x_1, \ldots, x_d)$ to output $Y = (y_1, \ldots, y_m)$ can be expressed as

$$Y = f(X) \tag{4.1}$$

Referring to Figure 1.5, the requirement is to find the set $\mathcal{R}_x$ in the input space $\mathcal{R}_p$ which places $Y = f(X)$ into the required performance or tolerance region $\mathcal{R}_Y$ in output space : $\mathcal{R}_q$. This is essentially an inversion problem and is sometimes referred to as inverse (or reverse) engineering : find

$$\mathcal{R}_x = \{X | Y = f(X) \in \mathcal{R}_Y\} = f^{-1}(\mathcal{R}_Y) \tag{4.2}$$

The methods consist of approximating $\mathcal{R}_x$ with say $\hat{\mathcal{R}}_x$ using observations $Y_i = f(S_i)$ at selected inputs. Thus these are also computer experiments but typically go directly to $\mathcal{R}_x$ rather than via an emulator. The methods often proceed sequentially by updating the 'estimated' region $\hat{\mathcal{R}}_x$ with the new inputs : $\hat{\mathcal{R}}_x(S_1, \ldots, S_n) = \hat{\mathcal{R}}_x^{(n)}$, say. Published work can be classified by the nature of $\hat{\mathcal{R}}_x^{(n)}$ and the updating rules $\hat{\mathcal{R}}_x^{(n)} \rightarrow \hat{\mathcal{R}}_x^{(n+1)}$. Also different conditions are required such as $\mathcal{R}_x \subseteq \hat{\mathcal{R}}_x^{(n)}$ or $\hat{\mathcal{R}}_x^{(n)} \subseteq \mathcal{R}_x$ or when $\hat{\mathcal{R}}_x^{(n)}$ is a single point in $\mathcal{R}_x$.

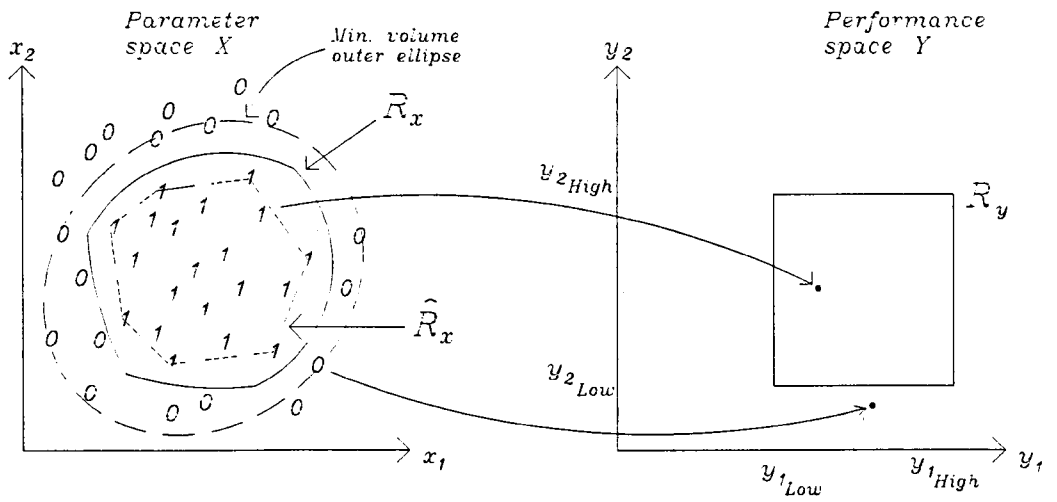


Figure 4.1: Estimation of $\mathcal{R}_x$ with a convex hull.

The type of information recorded and used which is similar to the specification of the updating rule $\hat{\mathcal{R}}_x^{(n)} \rightarrow \hat{\mathcal{R}}_x^{(n+1)}$ may also vary. At its simplest a method may only use a binary indicator

$$I_i = \begin{cases} 1 & Y_i = f(S_i) \in \mathcal{R}_Y, (i = 1, \ldots, n) \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

The good $S_i, (I_i = 1)$ can then be used to form $\hat{\mathcal{R}}_x^{(n)}$. For example one can form $\hat{\mathcal{R}}_x^{(n)}$ = convex hull of all good S_i so that if $\mathcal{R}_x$ is itself convex then $\hat{\mathcal{R}}_x^{(n)} \subseteq \mathcal{R}_x$. This is demonstrated in Figure 4.1 (compare with Figure 1.5) which shows $\mathcal{R}_x$, $\hat{\mathcal{R}}_x$ estimated with a convex hull and an ellipse bounding the region $\mathcal{R}_x$ for an example system $Y = f(X)$ with $X = (x_1, x_2)$ and $Y = (y_1, y_2)$. This relates directly to methods which estimate $\mathcal{R}_x$ with ellipsoids [3] and methods which extend this to the design centering problem by estimating the centre of $\mathcal{R}_x$, [2, 1].

We can use a more sensitivity based function as follows. Let $B(S_i)$ be some region (rectangle, hull) centred at S_i. Then it may be possible to find (or estimate) whether (i)

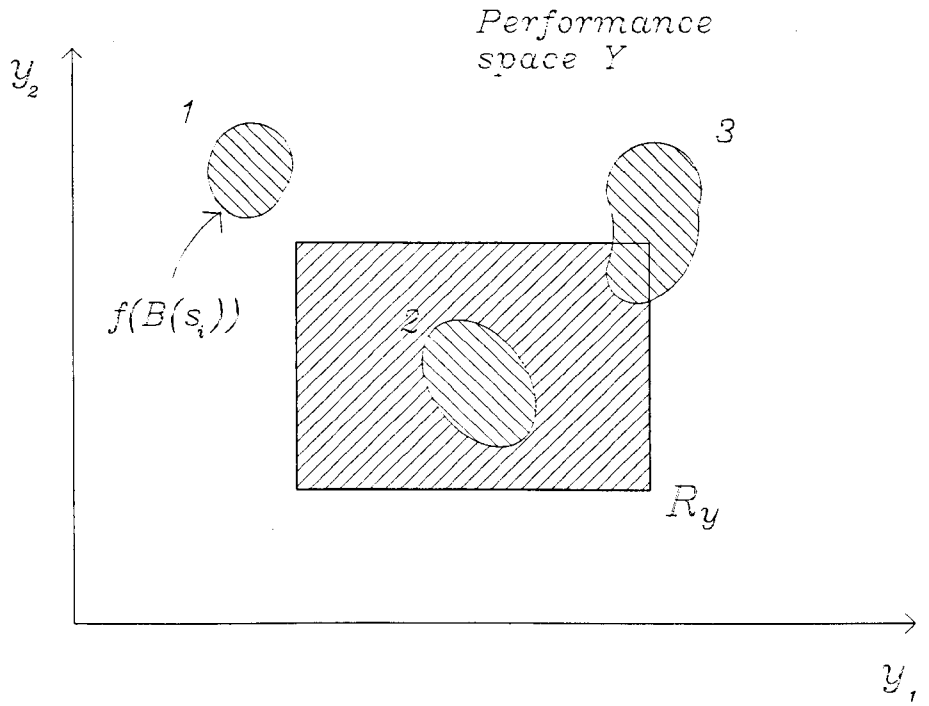


Figure 4.2: Three possible positions of the region B.

$f(B(S_i)) \subseteq \mathcal{R}_Y^c$ or (ii) $f(B(S_i)) \subseteq \mathcal{R}_Y$ or (iii) $f(B(S_i))$ overlaps the boundary of $\mathcal{R}_Y$. Figure 4.2 shows these three situations as 1, 2 and 3 respectively. Methods of estimating f^{-1} are described in [5, 4] which use interval arithmetic to approximate $\mathcal{R}_x$ by translating sets between parameter and performance spaces described in Section 1.2.3.

4.3 Optimization for robustness

Following the notation of Section 1.2 we consider for ease of presentation a system with two inputs $X = (x_1, x_2)$ and one output Y. Suppose following the performance region approach we require Y to lie in a region, defined as an interval, $\mathcal{R}_Y$. In addition assume that x_1 and x_2 are independent random variables with probability density functions

$p_1(x_1|\mu_1)$ and $p_2(x_2|\mu_2)$ where μ_1 and μ_2 are the means of x_1 and x_2 to be interpreted as nominal values. Following the "parameter design" ideas within RED we assume that μ_1 and μ_2 are controllable. The RED criteria, stated roughly is to keep $Y \in \mathcal{R}_Y$ while minimising the variation in Y and to do this through control of (μ_1, μ_2).

We deal first with the simple case when $\mathcal{R}_Y$ is a single target t. Then the mean squared error is given by

$$\text{MSE} = \text{E}(Y - t)^2 = \text{Var}(Y) + (\text{E}(Y) - t)^2 \tag{4.4}$$

where variances and expectations are with respect to the variation in x_1 and x_2. It is interesting to see what a classical sensitivity analysis gives. Thus expand Y in a Taylor expansion at (μ_1, μ_2) to obtain

$$Y(x_1, x_2) \approx Y(\mu_1, \mu_2) + (x_1 - \mu_1)\frac{\partial Y}{\partial x_1} + (x_2 - \mu_2)\frac{\partial Y}{\partial x_2} \tag{4.5}$$

where $\frac{\partial Y}{\partial x_1}$ and $\frac{\partial Y}{\partial x_2}$ are assumed to be evaluated at (μ_1, μ_2). This gives

$$\text{E}(Y) \approx Y(\mu_1, \mu_2) \tag{4.6}$$

and

$$\text{Var}(Y) \approx \sigma_1^2\left(\frac{\partial Y}{\partial x_1}\right)^2 + \sigma_2^2\left(\frac{\partial Y}{\partial x_2}\right)^2 \tag{4.7}$$

where σ_1^2 and σ_2^2 are the variances of x_1 and x_2 respectively. Then

$$\text{MSE} \approx \sigma_1^2\left(\frac{\partial Y}{\partial x_1}\right)^2 + \sigma_2^2\left(\frac{\partial Y}{\partial x_2}\right)^2 + (Y(\mu_1, \mu_2) - t)^2 \tag{4.8}$$

The approximate "unbiased" solution is to set

$$\text{minimise} \quad (\sigma_1^2\left(\frac{\partial Y}{\partial x_1}\right)^2 + \sigma_2^2\left(\frac{\partial Y}{\partial x_2}\right)^2) \tag{4.9}$$

subject to

$$Y(\mu_1, \mu_2) = t \tag{4.10}$$

We consider two ways of defining tolerances for system inputs:

(i) σ_1^2, σ_2^2 do not depend on μ_1, μ_2, termed the *absolute* tolerance case.

(ii) σ_1^2, σ_2^2 depend on μ_1, μ_2, termed the *relative* tolerance case.

In some branches of engineering it is common to specify a component value as $\mu \pm \delta\%$ corresponding to case (ii), whereas in areas such as mechanical engineering or manufacturing the specified tolerances could be absolute (see [6] for an example) i.e $\mu \pm \epsilon$ (case (i)). In the absolute tolerance case (i) we obtain a weighted measure of the flatness of the function $Y(x_1, x_2)$

$$\sigma_1^2\left(\frac{\partial Y}{\partial x_1}\right)^2 + \sigma_2^2\left(\frac{\partial Y}{\partial x_2}\right)^2 \tag{4.11}$$

and for the relative tolerance case (ii) we have $\sigma_1^2 \propto x_1$ and $\sigma_2^2 \propto x_2$.

In the situation where the output sensitivity is only affected by one input we can then minimise the sensitivity and use the second input to adjust to target, that is if neither $\frac{\partial Y}{\partial x_1}$ nor $\frac{\partial Y}{\partial x_2}$ depend on μ_2 then for any (σ_1^2, σ_2^2) we can solve the problem by moving μ_1 to where (4.11) is a minimum and correct to target by moving μ_2. It is worth exploring

the consequences of this latter condition. Thus suppose

$$\frac{\partial Y}{\partial x_1} = g(x_1) \; ; \quad \frac{\partial Y}{\partial x_2} = h(x_1) \tag{4.12}$$

The second equation here gives

$$Y = u(x_1) + x_2 h(x_1) \tag{4.13}$$

and substitution in the first yields $h(x_1) =$ a constant. Thus the general form is $Y = u(x_1) + ax_2$ that is linear in x_2 and additive across x_1 and x_2. This solution is independent of the (fixed for case (i)) values of σ_1 and σ_2.

In general, for a complex system, we will not have enough analytic information to perform optimization directly on the simulator. Even when the "sensitivities" $\frac{\partial Y}{\partial x_1}$ and $\frac{\partial Y}{\partial x_2}$ are available as output (see Section 1.2.3) these are still observables only and essentially add to the list of output factors.

The full unbiased solution which relates directly to the definition of quality in Section 1.1.1 is

$$\min \text{Var}(Y) \text{ subject to } E(Y) = t. \tag{4.14}$$

The alternative to analytic or approximate analytic solution is to estimate Var(Y) and E(Y) directly from output values for Y generated by a sample of input values. If $\sigma_Y^2 = \text{Var}(Y)$ and $\mu_Y = \text{E}(Y)$, we can call these estimates $\hat{\sigma}_Y^2$ and $\hat{\mu}_Y$ respectively. Then the solution is

$$\min \; \hat{\sigma}_Y^2 \text{ subject to } \hat{\mu}_Y = t. \tag{4.15}$$

Clearly as the control (μ_1, μ_2) changes we need to recompute new $\hat{\sigma}_Y^2$ and $\hat{\mu}_Y$.

The solution we propose here is a compromise between the inverse approach of (4.2) and the unbiased approach just described. Thus we assume rather than a simple target that $\mathcal{R}_Y$ is a target region for Y. Then we take as the problem

$$\min \ \sigma_Y^2 \ \text{subject to} \ \mu_Y \ \text{in} \ \mathcal{R}_Y. \tag{4.16}$$

We can express this using a penalty

$$\min \left(\sigma_Y^2 + \phi(\mu_Y)\right) \tag{4.17}$$

where

$$\phi(\mu_Y) = \begin{cases} 0 & \mu_Y \text{ in } \mathcal{R}_Y \\ \infty & \mu_Y \text{ not in } \mathcal{R}_Y \end{cases} \tag{4.18}$$

Now suppose as above we have estimates $\hat{\sigma}_Y^2$ and $\hat{\mu}_Y$ we shall simply use

$$\min \left(\hat{\sigma}_Y^2 + \phi(\hat{\mu}_Y)\right) \tag{4.19}$$

where again $\phi(.)$ is the penalty function for $\mathcal{R}_Y$. Of course by making $\mathcal{R}_Y = t$ we reduce to the simple target approach. A key point of the optimization is that all these operations are easily performed using a fast emulator of the simulator rather than the simulator directly.

4.4 The procedure

The estimates of σ_Y^2 and μ_Y are given by generating sample points (x_{i1}, x_{i2}), $(i = 1, \ldots, n)$ and estimating by

$$\hat{\mu}_Y = \frac{1}{n} \sum_{i=1}^{n} Y(x_{i1}, x_{i2}) \tag{4.20}$$

$$\hat{\sigma}_Y^2 = \frac{1}{n-1} \sum_{i=1}^{n} (Y(x_{i1}, x_{i2}) - \hat{\mu}_Y)^2 \tag{4.21}$$

We use two methods of generating (x_{i1}, x_{i2}) : (i) simple Monte Carlo sampling for x_1 and x_2 and (ii) a method based on low-discrepancy integer lattices described in Section 2.3.1. If $F_j(x_j)$ is the cumulative distribution function of x_j, $(j = 1, 2)$ and u_{ij} $(i = 1, \ldots, n)$ is an independent Monte Carlo sample from a uniform distribution in $[0, 1]$ $(j = 1, 2)$, then

$$x_{ij} = F_j^{-1}(u_{ij}) \quad (i = 1, \ldots, n, \; j = 1, 2) \tag{4.22}$$

We generate an integer lattice [7, 8] on the square $\otimes^2[0, 1]$ based on a single integer generator (g_1, g_2) as in Section 2.3.1 and use the same transform in (4.22) to mimic the distribution of x_1 and x_2.

The distributions F_j are changed as the control (μ_1, μ_2) is changed. Thus in the Gaussian case $X_j \sim N(\mu_j, \sigma_j)$ we simply take the x_{ij} as a standard N(0, 1) sample and transform

$$\tilde{x}_{ij} = \sigma_j x_{ij} + \mu_j \tag{4.23}$$

Clearly this is possible for any shift-scale family. This means that we need only generate

a single Monte Carlo sample or lattice.

Having found the estimates (4.20),(4.21) using either the Monte Carlo or the Lattice methods these are then used in (4.19) together with the global optimizer described in Section 2.5. All the above material can be extended in a straightforward way to to higher dimensional input spaces and, indeed, we shall use a six-dimensional example as a case in the next section.

4.5 Case study

4.5.1 Introduction

We continue the analysis of the voltage amplifier circuit described in Chapter 3, Section 3.4 and illustrated in Figure 3.2. For an analytical study of the circuit we assume the operational amplifier to be ideal and, assuming DC conditions, the circuit can be further simplified by (i) setting all capacitors to open circuit, (ii) assuming no load on the output (pin 'Vout' in Figure 3.2) and (iii) setting $V_{cen} = 0$. An analysis of the circuit yields the equation

$$V_{out} = \frac{(R_4 + R_5 + R_6)(R_3 V_{in} + R_1 V_{cen} + R_2 V_{cen})}{(R_1 + R_2 + R_3)(R_4 + R_5)} \tag{4.24}$$

where V_{cen} is the offset voltage. Other design constraints can be introduced to further simplify the analysis as follows. If we set $R_a = R_1 = R_2 = R_4 = R_5$ and $R_b = R_3 = R_6$ as in the nominal design (4.24) can be re-written as

$$V_{out} = V_{in}\frac{R_b}{2R_a} + V_{cen} \tag{4.25}$$

This constraint is used for the initial design where the nominal setting of circuit parameters $R_a = 20\text{k}\Omega$, $R_b = 270\text{k}\Omega$ yields $V_{out} = 3.864$.

The performance of the circuit is summarised in Table 3.3. The goal of the optimization process which follows is to minimize the variance of the circuit for the tolerance levels set in Section 3.4, given a target interval $\mathcal{R}_Y$ for the response.

4.5.2 Experimentation

Continuing from Section 3.4 a Robust Design experiment is carried out using the RCD software module. Instead of using the regression analysis in the RCD package a DACE model is fitted to the experimental results. For experimentation the circuit parameters need to be varied over a suitable range which defines the region over which they will be optimized, $\mathcal{R}_p$. The 12 components are thus selected at the given nominal values with an input space of $\pm 40\%$ of their nominal values. Using a Latin Hypercube design with 50 runs an experiment is conducted to produce the DACE model of Figure 4.3. This is then used to construct the main effects plots of Figure 4.4 which show that the variation in response is due to the six resistors $R_1, \ldots, R_6$ which correspond to the six resistors in equation 4.24. The plots show how changes in circuit parameters affect response and can act as a guide to optimization by hand as well as displaying the main causes of response variation.

The DACE model is used as an emulator of the simulator to predict circuit response as part of a global optimization procedure.

```
MLE RESULTS

The Response Variable is @voltage

N= 50 NX= 12 THE COVARIANCE INDEX= 1
SIGMAZ= 5.2472e+01 -2*LN LIKELIHOOD= -1.9808e+02

NUMBER OF LINEAR MODEL PARAMETERS IS: 1
Variable                  Beta        Std. Err.     t-val
-------------------------------------
Constant                6.6790e+00   0.0000e+00          Inf

GAMMA=   0.0000
THETA=  5.0117e-06 1.0435e-05 1.8244e-14 1.0142e-07 5.7943e-02
THETA=  5.8526e-02 6.6602e-02 1.3969e-01 8.6565e-03 3.6964e-03
THETA=  2.2541e-06 1.0109e-08
POWER=  1.0054e+00 1.6535e+00 1.6820e+00 1.9253e+00 2.0000e+00
POWER=  2.0000e+00 1.9990e+00 1.9965e+00 2.0000e+00 2.0000e+00
POWER=  1.0101e+00 1.7581e+00

TIME (MIN.) FOR LIKELIHOOD CALCULATIONS IS:    7.13
THE DATA FOR THIS RUN IS IN THE FILE dace.x AND dace.y
THE DATA WERE TRANSLATED TO [-0.5,0.5] FOR THIS ANALYSIS
```

Figure 4.3: DACE model for voltage amplifier circuit

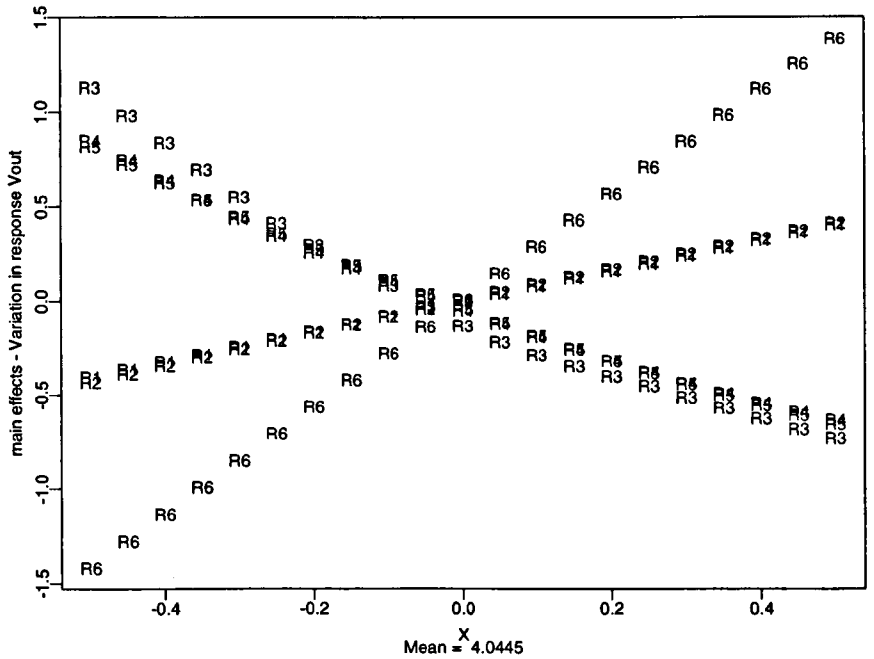


Figure 4.4: Main Effects plots for DACE model

4.5.3 Analytical optimization

To provide insight into the optimization procedure we use the simplified system equation (4.24) to carry out the analytic method in (4.9) and (4.10) using two factors R_a and R_b.

$$\mathrm{Var}(V_{out}) = \sigma_a^2 \left(\frac{-V_{in} R_b}{2R_a^2}\right)^2 + \sigma_b^2 \left(\frac{V_{in}}{2R_a}\right)^2 \tag{4.26}$$

Setting $(V_{in} = 0.2, V_{cen} = 2.5, V_{out} = 3.86)$ we obtain a target constraint of $R_b = 13.6R_a$. We relate $\sigma_a = \mathrm{SD}(R_a)$ and $\sigma_b = \mathrm{SD}(R_b)$ in two ways corresponding to absolute and relative tolerances respectively (Section 4.3):

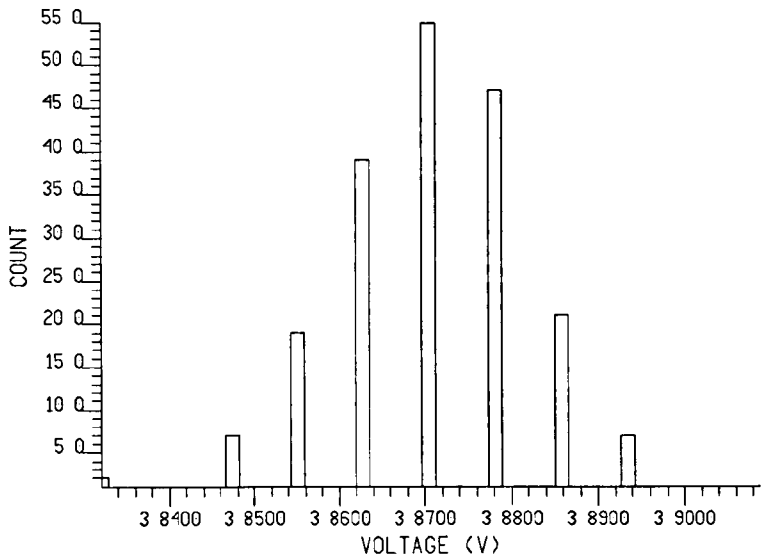


Figure 4.5: Histogram of voltage output for analytically optimized design

Case (i) : absolute tolerances.

In this case:

$$\sigma_b = (R_b/R_a)\sigma_a \tag{4.27}$$

Combining (4.27) with the target constraint $R_b = 13.6R_a$ the standard deviation of V_{out} is represented by

$$\mathrm{SD}(V_{out}) = 1.923\frac{\sigma_a}{R_a}. \tag{4.28}$$

Thus the simple solution is to maximize R_a within the defined space yielding the solution $(R_a, R_b) = (28, 380)$ kΩ. This gives a decrease in $\mathrm{SD}(V_{out})$ from $1.665 \times 10^{-4}\sigma_a$ to $1.189 \times 10^{-4}\sigma_a$ a reduction of 28.5%. The confirmation of this design with a 200 run Monte Carlo simulation is given in Table 4.1 with a histogram in Figure 4.5.

Case (ii) : relative tolerances.

$$\sigma_a = cR_a,\ \sigma_b = cR_b \tag{4.29}$$

where c is a constant. In this case the standard deviation of V_{out} is represented by

$$\mathrm{SD}(V_{out}) = 0.1414c\frac{R_b}{R_a}. \tag{4.30}$$

which, when combined with the target constraint $R_b = 13.6R_a$, shows that the variation of the target is, at least approximately, independent of the nominal values. This implies that in the relative tolerance case the circuit is already stable and we shall not perform optimization in this case.

4.5.4 Global circuit optimization

Because the DACE model emulator can be evaluated many times faster than the circuit simulator it can be used in conjunction with the global optimization algorithm to improve the circuit design according to the criterion in (4.16), that is min σ_Y^2 subject to μ_Y in $\mathcal{R}_Y$. We choose $\mathcal{R}_Y = [3.80, 3.92]$ and adopt the penalty function strategy (4.17) where:

$$\sigma_Y^2 = \begin{cases} \hat{\sigma}_Y^2 & \mu_Y \text{ in } \mathcal{R}_Y \\ 5 \times 10^{-4} & \mu_Y \text{ not in } \mathcal{R}_Y \end{cases} \tag{4.31}$$

The value 5×10^{-4} is chosen as an average value for $\hat{\sigma}_Y^2$ during the optimization. The final circuit design is to set the six resistors $R_1, \ldots, R_6$ to an *absolute* tolerance of $\pm 1\%$

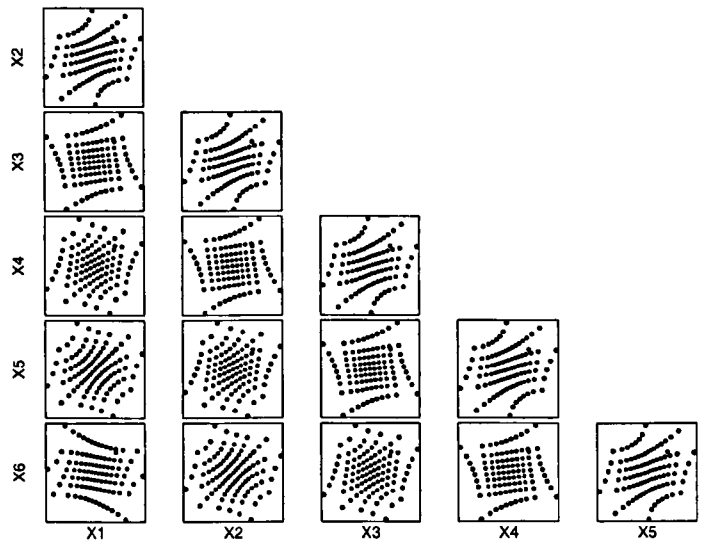


Figure 4.6: Gaussian lattice for estimating μ and σ

of the original nominal values giving $(R_3, R_6) \pm 2.7\text{k}\Omega$ and $(R_1, R_2, R_4, R_5) \pm 0.2\text{k}\Omega$, the rest $\pm 20\%$ of the original nominal values. Because of their lack of significance we maintain the values of parameters $C_1, \ldots, C_4, R_7, R_8$ at their nominal values and only vary the others when predicting with the emulator. Both lattice and Monte Carlo methods of calculating the estimates $\hat{\sigma}_Y^2$ and $\hat{\mu}_Y$ are compared in different optimizations of the design. At each design point selected by the optimizer the circuit is emulated at $n = 100$ points according to either lattice or Monte Carlo distributions centered at the selected nominals with the appropriate scaling. The lattice and Monte Carlo points chosen for this example in six dimensions can be seen as pairwise plots in Figures 4.6 and 4.7. Once optimized the circuit designs are confirmed on the simulator by a 200 run Monte Carlo analysis which give the histograms of Figures 4.8 and 4.9. The results are summarised in Table 4.1 and show that the optimizer has found solutions (confirmed by a 200 run Monte Carlo simulation using the simulator) which show improvements for

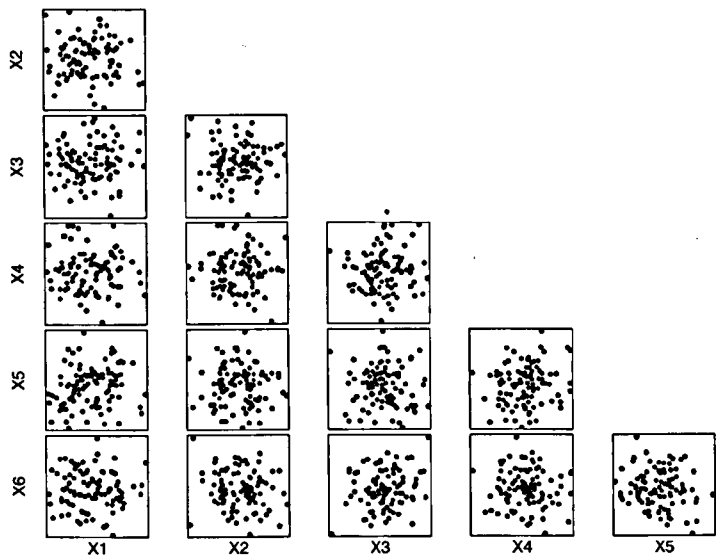


Figure 4.7: Gaussian Monte Carlo sample for estimating μ and σ

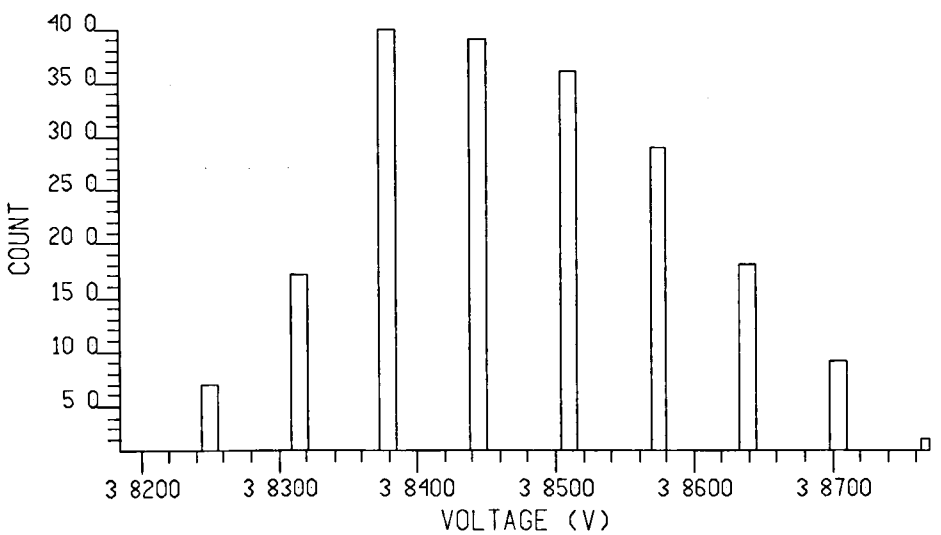


Figure 4.8: Histogram of voltage output for lattice-optimized design

	Initial design using simulator	Analytic optimization	Simulator Confirmation	
			Lattice	Monte Carlo
Mean, $\hat{\mu}$	3.86	3.87	3.84	3.82
Var, $\hat{\sigma}^2$	2.85e-4	1.36e-4	1.46e-4	1.60e-4

Table 4.1: Original design, optimized designs and Monte Carlo confirmatory experiments, absolute tolerance case.

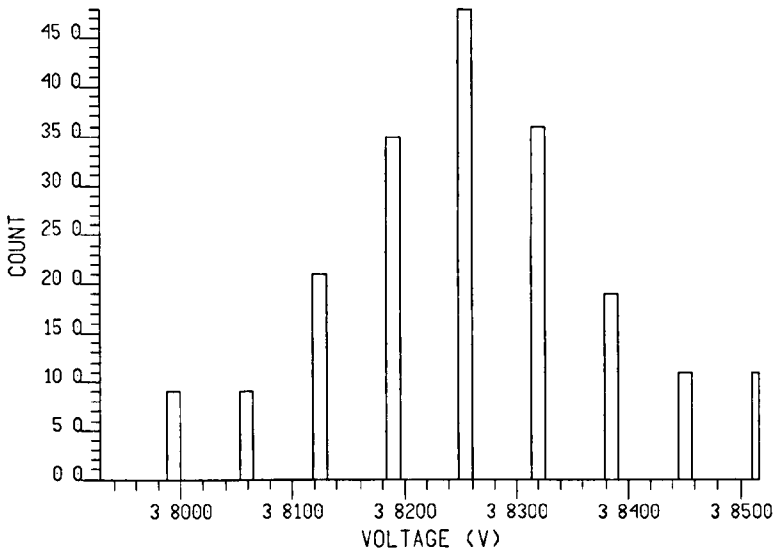


Figure 4.9: Histogram of voltage output for Monte Carlo-optimized design

$\mathrm{Var}(V_{out})$ of the initial design by 48.7% for the Lattice approach and 43.8% for Monte Carlo. It is interesting to note that the simple analytic method actually yields the best results with an improvement of 52.2%. The point here is that in a larger and more complex circuit such an approach is impractical. The optimizer parameters were set to observe 500 points in $\mathcal{R}_p$ (each point involving estimating μ_Y and σ_Y^2 with 100 evaluations of the DACE model) with 50 iterations taking $6\frac{3}{4}$ hours to find a solution using a Sun SparcStation2. An equivalent number of evaluations using the simulator directly would take approximately 1600 hours. The discrepancy in values between the optimizer (using the DACE model) and the simulator may be explained by (i) the emulator accuracy of around 2% when calculating $\hat{\mu}_Y$ and (ii) the optimizer estimates being based on a sample size of 100 points compared with 200 points for the Monte Carlo confirmations using the simulator.

The results given by the emulator from the optimization using the lattice estimator are

Component	Initial design		Lattice optimization		Monte Carlo optimization	
	Nominal	Tolerance	Nominal	Tolerance	Nominal	Tolerance
R1	20kΩ	±200Ω	27.4kΩ	±200Ω	26.6kΩ	±200Ω
R2	20kΩ	±200Ω	14.8kΩ	±200Ω	17.3kΩ	±200Ω
R3	270kΩ	±2.7kΩ	370kΩ	±2.7kΩ	313kΩ	±2.7kΩ
R4	20kΩ	±200Ω	27.7kΩ	±200Ω	27.8kΩ	±200Ω
R5	20kΩ	±200Ω	24.2kΩ	±200Ω	26.5kΩ	±200Ω
R6	270kΩ	±2.7kΩ	314kΩ	±2.7kΩ	375kΩ	±2.7kΩ

Table 4.2: Parameter values before and after optimization

$\mu = 3.80$, $\hat{\sigma}^2 = 1.71e-4$ and using the Monte Carlo estimator $\mu = 3.80$, $\hat{\sigma}^2 = 1.29e-4$. The parameter values chosen for components $R_1, \ldots, R_6$ are given in Table 4.2. Note that the optimized design values of $\hat{\mu}$ are at the lower bound of the target interval $\mathcal{R}_Y = [3.80, 3.92]$. Returning to the much simplified circuit analysis resulting in equation 4.25 we display the surface of this function as $V_{out} = f(R_a, R_b)$ over the optimization region $\mathcal{R}_p$ in Figure 4.10. This shows that, although the surface is derived from a much simplified version of the real function, decreasing the target response value places the circuit response in a flatter area which results in less variation in response for a given absolute parameter variation. In the relative variation case this is counteracted by the increase of σ_a with R_a.

In modelling the whole function, rather than a simplified version, the emulator is a truer representation of the system and has more freedom in finding an optimal solution. From looking at Table 4.2 one can see the results using the emulator give parameter values different to those obtained by the analytic optimization indicating the difference between the simplified mathematical model (4.25) and the DACE model emulator.

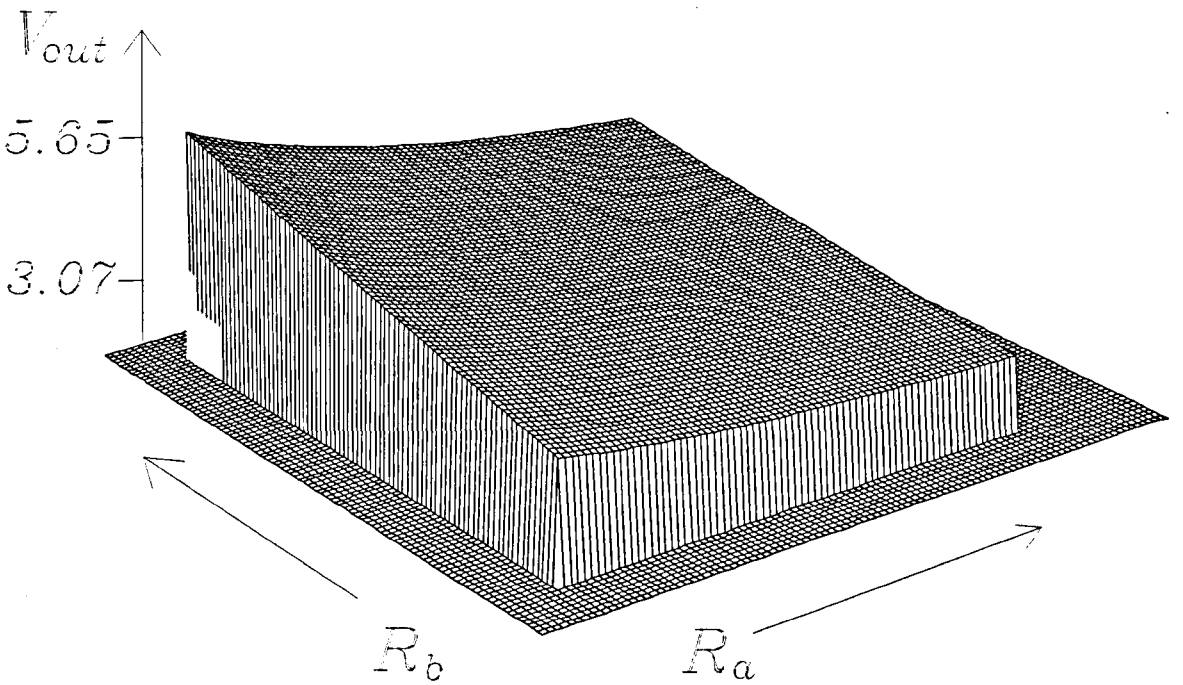


Figure 4.10: Surface of simplified response function over region $\mathcal{R}_p$.

4.6 Conclusions

This chapter describes a method of design optimization with respect to quality as defined in Section 1.1.1. The method presented is defined in terms of parameter and performance space following closely work on design centering and tolerancing but takes advantage of the concept of emulation to improve the efficiency of optimization allowing the use of a numerical optimizer. The case study, which follows directly from the study in Chapter 3, describes the approach and shows an improvement over the initial design by reducing response variability by 48.7%. This is achieved in under 7 hours using the emulator, an equivalent number of calculations on the simulator being estimated at 1600 hours. Although the example system has only 12 input factors the method presented

represents a vast improvement over traditional Monte Carlo methods and provides the opportunity for global rather than local design optimization. The optimization method is directly applicable to problems in higher dimensions and the following Chapters describe techniques for reducing the complexity of building emulators for such higher-dimensional systems for optimization. An important distinction is between relative and absolute tolerance settings for system parameters and the differences are explored in the case study. The global optimization solution does well compared with the solution obtained analytically showing the validity of the approach.

References

[1] H L Abdel-Malek. The ellipsoidal technique for design centering and region approximation. *IEEE Trans. Comp. Aided Des.*, 10:1006–1013, Aug 1991.

[2] R K Brayton and R Spence. *Sensitivity and optimization.* Elsevier, Amsterdam, 1980.

[3] S W Director and G D Hachtel. The simplicial approximation approach to design centering. *IEEE Trans. Circuits & Systs.*, CAS-24:363–372, July 1977.

[4] L Jaulin and E Walter. Guaranteed nonlinear parameter-estimation from bounded-error data via interval-analysis. *Mathematics and Computers in Simulation*, 35(2):123–137, 1993.

[5] L Jaulin and E Walter. Set inversion via interval-analysis for nonlinear bounded-error estimation. *Automatica*, 29(4):1053–1064, 1993.

[6] Peter Mucci. *Handbook for engineering design using standard materials and components.* P E R Mucci Ltd., The Old Bakery, Parsonage Lane, Durley, Southampton SO3 2AD, 1986.

[7] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* CBMS-NFS, SIAM, Philadelphia, 1992.

[8] Fang K T and Wang Y. *Number-theoretic Methods in Statistics.* Chapman & Hall, London, 1994.

[9] Anatoly A. Zhigljavsky. *Theory of Global Random Search*, chapter 4. Kluwer Academic Publishers, 1991.

Chapter 5

Robust circuit design II: Decomposition of complex systems

5.1 Introduction

Part of the difficulty of performing Robust Engineering Design on large systems is the execution of the experiment. When using computer simulators large system models can be costly to compute. The rationale for this chapter is that partitioning a system into individual subsystems for analysis will increase the ability to build emulators of complex systems for optimization. Decomposition is achieved using the partitioning algorithm detailed in Section 2.4.1. Emulator models of each sub-system are derived according to the method of Section 2.3.2 and are combined to emulate the full system as part of the RED strategy described in Chapter 3.

The main issue is one of *preserving the environment in which the sub-systems exist*, this is achieved using small circuit blocks to mimic the effect of connecting the sub-circuits together, hereby referred to as *load blocks*. Sub-sections of complex designs can be analysed independently with the aim of making simulation and emulator model building more efficient and accurate. The approach presented here relates to a design being decomposed in a linear fashion for piecewise analysis with feedback between sub-sections expressed via the load blocks. The outline of the proposed method is

i. Form a graph representing the circuit.

ii. Use a partitioning algorithm to decompose the graph.

iii. Formulate sub-circuits according to the decomposed graph.

iv. At the sub-circuit boundaries add a load block to mimic the missing connections.

v. Build sub-emulators of each sub-circuit.

vi. Combine the sub-emulators to emulate the whole circuit.

vii. Use the emulator to optimize the design.

The generalisation of the methods to multi-way partitions possibly including feedback is an area for possible future research.

5.1.1 Simulation

There are two basic options for the analysis of electronic circuits using SPICE-based simulators:

i. AC analysis, for simulation in the frequency domain, and,

ii. Transient analysis, for simulation in the time domain.

Each method is used to measure different aspects of circuit response. From this point of view, the choice of AC or Transient Analysis affects only the setup of the simulator and the set of responses which can be measured as the same type of empirical model is fitted. However when decomposing a circuit the type of analysis selected defines the way in which the load blocks are modelled and how the sub-emulators are combined to form an emulator of the whole circuit. The method outlined above is applied to both AC and Transient analysis.

In this way it is proposed to extend basic RED methodology to the analysis of complex circuits and systems.

5.2 Circuit description

The circuit of Figure 5.3, an audio pre-amplifier circuit, is used in this chapter to demonstrate the ideas presented. The main function of the circuit is to convert several different transducer signals to a signal appropriate for input to an audio amplifier. The circuit therefore needs to cope with a wide range of inputs and provide suitable biasing of the signal to account for non-linearities present in the transducers. The study concentrates on measuring the response at the pin V_{out} due to a signal input to the magnetic pickup ('Mag_PU' in Figure 6.1).

5.3 Partitioning

Circuit partitioning algorithms are used in VLSI design where circuits too large to be placed on one chip are split between several chips [7]. The requirement is to partition the circuit to minimize the number of connections between blocks. We adopt a similar approach using the circuit topology in a simple graph-theoretic way. The partitioning algorithm described in Section 2.4.1 is used as the basis for an improved algorithm which is then used to decompose a circuit graph into three separate sections. Following the description of the basic partitioning algorithm described in Section 2.4.1, the improved algorithm is now described and tested.

5.3.1 An improved partitioning algorithm

Using notation from Section 2.4.1 we note that in point (ii.) of Section 2.4.1 it is possible to be in a tie situation when choosing a cell to move i.e having more than one candidate for the base cell, c^b. The algorithm has been improved to deal with this situation. Instead of arbitrarily choosing c^b we choose the cell with the least number of connections to other cells in its current block and if there is still a tie the most number of connections to cells in the complementary block. We define two counters $\mathcal{F}$ and $\mathcal{T}$ for a given base cell c^b such that

i. $\mathcal{F}(c^b)$ is the number of edges that cell (c^b) is connected to in the block it is moving *from.*

ii. $\mathcal{T}(c^b)$ is the number of edges that cell (c^b) is connected to in the block it is moving *to.*

Graph	Basic FM		Improved FM	
	mean passes	mean cut nets	mean passes	mean cut nets
Random	6.08	10.0	4.16	9.48
Geometric	5.32	2.4	4.48	1.36

Table 5.1: Random graph results table

This is in place of choosing the cell with the best balance ratio. Considering a set of n possible base cells, $c_1^b, \ldots, c_n^b$, we choose the cell having

$$\min\left(\mathcal{F}(c_i^b)\right), \qquad i = 1, \ldots, n \tag{5.1}$$

breaking further ties by choosing

$$\max\left(\mathcal{T}(c_i^b)\right), \qquad i = 1, \ldots, n. \tag{5.2}$$

The improvement can be incorporated as a natural extension of the algorithm as $\mathcal{F}(n_i)$ and $\mathcal{T}(n_i)$ are computed in order to establish *critical* nets (see above for a definition of a critical net). The modification leads to better grouping of cells in a tie situation choosing c^b with the least number of connections in block $\mathcal{F}$ and the most in block $\mathcal{T}$. The results of this modification can be seen in Table 5.1 where the improved FM algorithm is tested against the original algorithm. The Table shows the results of generating 100 graphs and decomposing each one using both the original and improved algorithms. The algorithm starts with an initial random partition and stops when no further improvement is made on minimising the size of the cut-set. The mean number of passes is the number of passes the algorithm made before finding an optimal partition for each graph divided by the number of graphs tested. The mean number of cut nets is the size of the best cut set

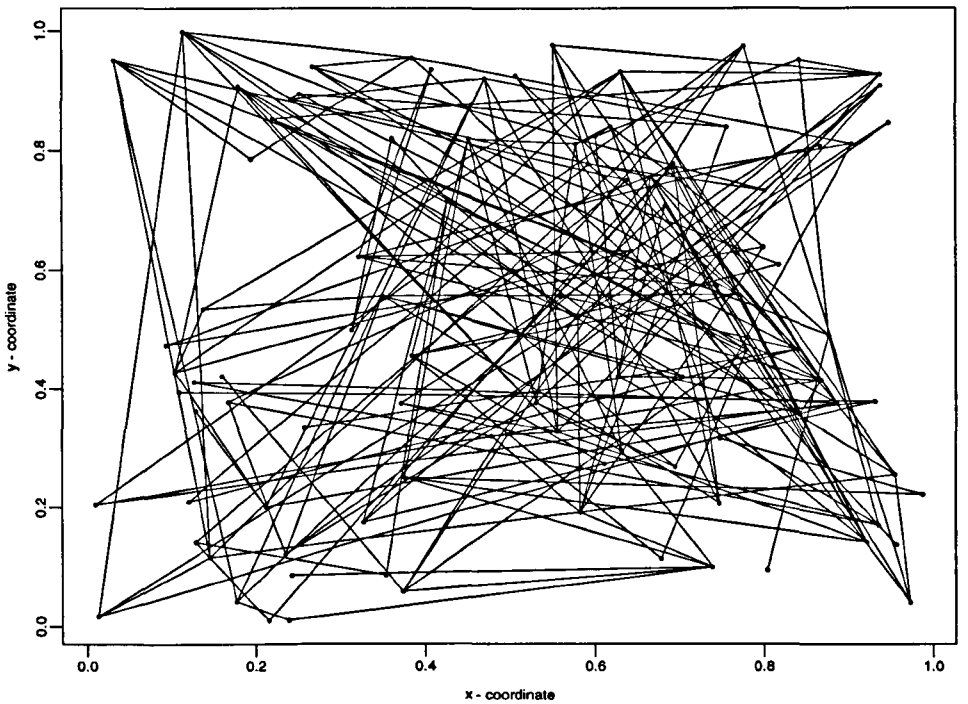


Figure 5.1: Example random graph: $n = 100$, $d = 4$

found by the algorithm for each graph divided by the number of graphs tested. The modified FM algorithm gives better solutions to the min-cut problem for almost no extra computational effort. Two types of graph are used as benchmarks, random graphs and geometric graphs both are described below.

Random graphs

For a graph of n nodes the probability p_r that any pair of nodes are connected is given as

$$p_r = \frac{d}{n-1} \tag{5.3}$$

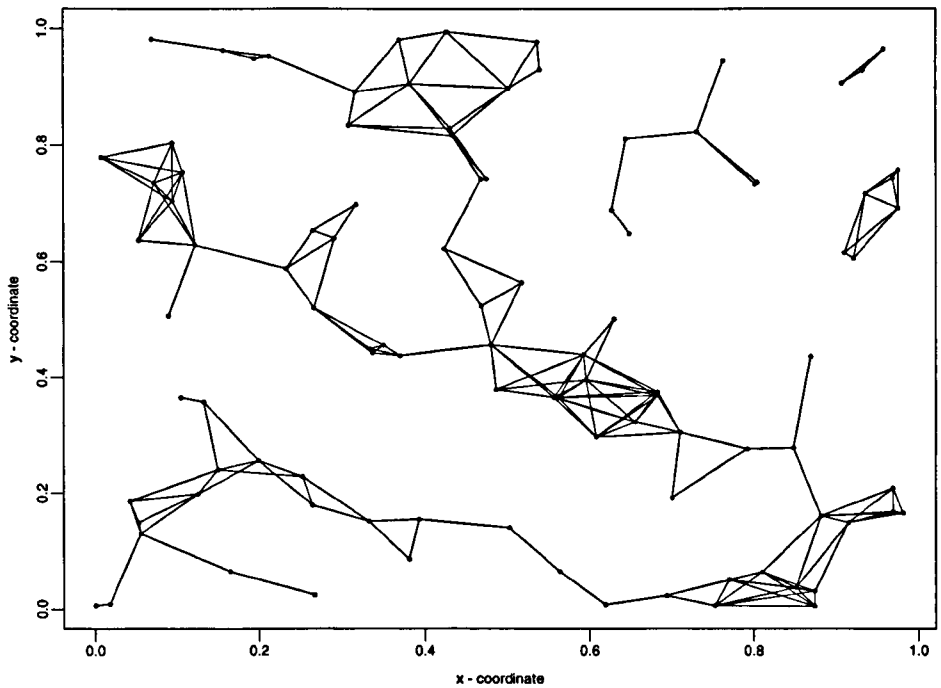


Figure 5.2: Example geometric graph: $n = 100$, $d = 4$

where d is the required degree of each node. Nodes unconnected after this procedure are assigned a single edge with another node chosen randomly to ensure that all nodes are connected to at least one other node, i.e $d_{min} = 1$, otherwise they do not form part of the network. To test the algorithms the values $n = 100$, $d = 4$ were chosen, see Figure 5.1 for an example. The graphs are constructed using the 'C' program 'listgen.c' in Appendix A.3.

Geometric graphs

For n nodes randomly placed in the space $[0, 1]^2$ the number of edges e of the graph is determined by

$$e = \frac{\sum_{i=1}^{n} d_i}{2} \tag{5.4}$$

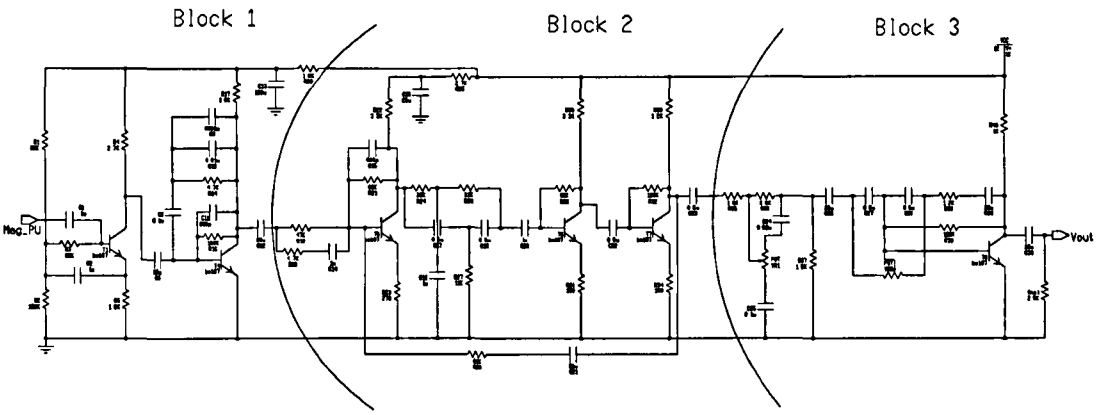


Figure 5.3: PA20 circuit - partition points

and

$$d = \frac{\sum_{i=1}^{n} d_i}{n} \tag{5.5}$$

to give

$$e = \frac{nd}{2} \tag{5.6}$$

where d is the required average degree as before. Unconnected nodes are then connected via a single edge to the nearest node to ensure $d_{min} = 1$ as for Random Graphs. To test the algorithms the values $n = 100$, $d = 4$ were chosen, see Figure 5.2 for an example graph. The graphs are constructed using the function 'mkgraph' defined in Appendix B.1 using the statistical package 'S-plus'.

5.3.2 Partitioning the circuit

The circuit and resultant partitions are shown in Figure 5.3. By relaxing the balance tolerance the algorithm was able to choose an initial partition of one small section and one larger section, the results of which are shown in Table 5.2 The balance tolerance was

Balance ratio (0 – 50) = 25		
Pass	Best move	No. of cut nets
1	#31	6
2	#25	5
3	#1	3
4	#69	3

Table 5.2: Output of MinCut algorithm for first partition

Balance ratio (0 – 50) = 10		
Pass	Best move	No. of cut nets
1	#23	8
2	#26	4
3	#2	3
4	#49	3

Table 5.3: Output of MinCut algorithm for second partition

then tightened and the larger section bisected, see Table 5.3.

5.4 The load blocks

Decomposition into sub-circuits affects overall circuit response. The response of the whole circuit is not equivalent to the sum of responses of individual sub-circuits because the interaction between sub-circuits is lost when considered individually. To overcome this difficulty the interaction between sub-circuits is accounted for by considering the effect of connecting one sub-circuit to another. The Substitution Theorem [4] is used to preserve the original network, it states that

> If any part of a network is replaced by any other combination of elements such that the terminal conditions remain unaltered, the conditions within the remainder of the network will remain unchanged.

Sub-circuits connected to the part of the circuit of interest can be considered as *load* impedances at the connection point. Considering the system of Figure 5.10 the load placed on the output of Block 1 is the input impedance of Block 2. Once this is found the 'loading effect' (L_1) of connecting Block 2 to Block 1 can be accounted for and Block 2 can be replaced by a much-simplified load block. The process of finding the correct loading factor for both transient analysis and AC Analysis is described for this particular form of decomposition.

The first step in defining a load block is to calculate the input impedance ($\mathcal{Z}$in) of the sub-circuit to be modelled. The simulator can be used for this purpose to calculate $\mathcal{Z}$in as a function of frequency. By performing an AC analysis over a suitable frequency range, with an additional current-sensing resistor at the input to the circuit, the input voltage and current can be measured. From this $\mathcal{Z}$in can be deduced for the stated frequency range as a complex function of frequency. This can then be interpreted according to the type of analysis to be performed. Figure 5.4 shows the process.

5.4.1 AC load blocks

For AC Analysis the input impedance $\mathcal{Z}$ is a function of frequency and is therefore represented by the function $\mathcal{Z}(\omega)$. $\mathcal{Z}(\omega)$ is approximated by System Modelling Blocks (SMB's) which are available in the Mentor Graphics version of SPICE used here and also in SPICE 3E2. The function is of the form:

$$\mathcal{Z}(\omega) = \frac{k(j\omega + z_1)(j\omega + z_2)}{(j\omega + p_1)(j\omega + p_2)} \tag{5.7}$$

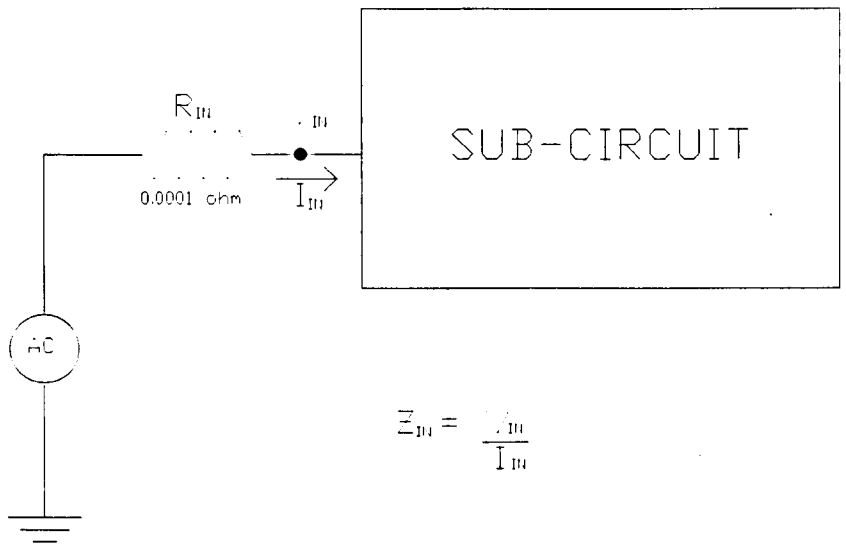


Figure 5.4: Finding $\mathcal{Z}(\omega)$ of a sub-circuit using AC analysis.

where ω is the frequency in radians. A global optimization algorithm (see Section 2.5) is used to find the parameters z_1, z_2, p_1, p_2 which give the best fit of the model to $\mathcal{Z}(\omega)$. A second-order model is assumed. If the results of the optimization process are poor a higher order model can be fitted. These parameters are used in the SMB's to represent the loading effect of the missing circuit, the circuit diagram for block 1 of the PA20 circuit, including the SMB, is shown in Figure 5.5.

The input impedance $\mathcal{Z}(\omega)$ is modelled with the SMB's in the following way.

i. Measure I_{in}, the input current of the sub-circuit to be modelled. For an AC analysis with an input voltage of 1v this is equivalent to $\frac{1}{\mathcal{Z}}$ from Ohm's law.

ii. Starting with a 2nd order function for the SMB find the parameters which fit the function $\frac{1}{\mathcal{Z}}$. If no model can be fitted use a higher order function.

iii. Use the SMB to sense the voltage at the output pin of the sub-circuit to which the

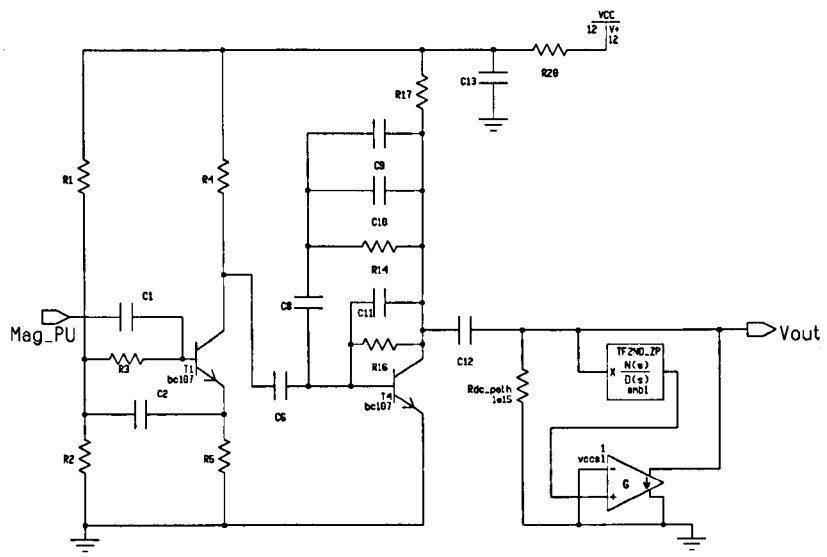


Figure 5.5: Block 1 of pa20 circuit

load block is to be attached.

iv. From the voltage output of the SMB create the relevant current at the sub-circuit output with a voltage-controlled voltage source (VCVS).

The load block can be seen in place in Figure 5.5. Note that, for the example, only the magnitude part of the complex current I_{in} was fitted with the optimizer. For phase-sensitive applications the optimizer may be required to fit both magnitude and phase.

5.4.2 Transient load blocks

For Transient Analysis the circuit is simulated with an input signal of a particular frequency. The input impedance of a sub-circuit can be represented as a complex number $x + jy$ for that frequency. This complex impedance can be represented by the

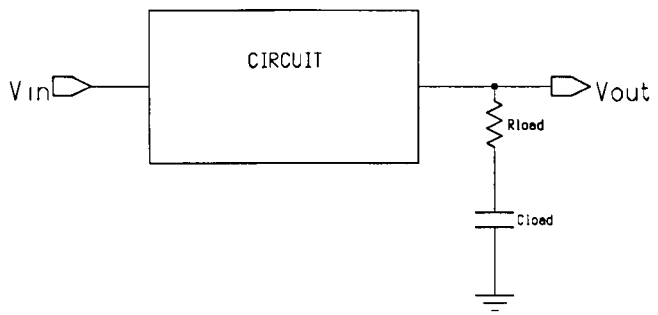


Figure 5.6: RC load

simulator as a resistor, R_L, and capacitor, C_L, in series (Figure 5.6), where:

$$R_L = x \tag{5.8}$$

$$C_L = \frac{-1}{\omega y} \tag{5.9}$$

Following the steps at the beginning of this section gives graphs of how the phase and magnitude of the complex $\mathcal{Z}in$ varies with frequency. Given this graph $\mathcal{Z}in$ can be deduced for the frequency of interest.

5.5 Robust Circuit Design experiments

5.5.1 AC analysis

The graph of Figure 5.7 shows the function $\mathcal{Z}(\omega)$ as measured by the simulator with the results of fitting equation 5.7 with the optimizer. Table 5.4 shows the parameter values for SMB1 and SMB2 of the circuit, Figure 5.10.

AC analysis is carried out in the following way:

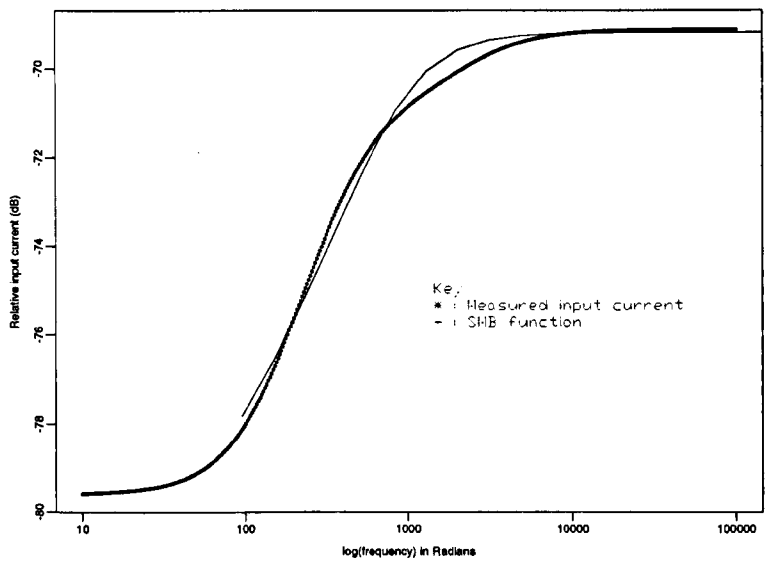


Figure 5.7: Fitting SMB model parameters to $\mathcal{Z}(\omega)$

	P1	P2	Z1	Z2	K
SMB 1	59.84	40.85	69.68	1.19e-2	2.24e-4
SMB 2	158.6	714.9	95.55	353.6	3.47e-4

Table 5.4: Parameters for SMB load blocks

1. Perform an experiment with input factors set to nominal for each block required to be modelled with a SMB.

2. Find parameters for the SMB's for each block in 1.

3. Experiment on each block measuring the relevant response(s).

The SMB parameters are not included in the RCD experiment as they will not influence the emulator model unless interactions between sub-circuits are expected. This is seen in the results of the RCD experiment on the PA20 circuit where the SMB parameters are included in the analysis. Only the parameter K shows up in the main effects plots (characterised by the letter 'T') of Figure 5.8 for the response at 200Hz (response number 6), its effect being very small compared with the effects of the other factors (note: each factor plot is scaled individually). Main effects plots are described in Chapter 3.
Section 5.5.3 describes how to include SMB parameters in the RCD experiment.
The experiments and DACE modelling were carried out using AC Analysis on the full PA20 circuit as well as for the three blocks resulting from its decomposition. The AC response of the circuit can be seen in Figure 5.9 which also shows the 12 points on the curve at which the circuit response is measured.
The general scheme for analysis can be seen in Figure 5.10. The results of the analysis of the full PA20 circuit are compared with those obtained from partitioning the system. Table 5.5 shows the number of input variables and simulations for the partitioned circuit experiment and the whole circuit experiment.
Three of the 12 points along the response curve of Figure 5.9 were chosen for analysis. A more comprehensive approach is to model the whole curve as a single response. Such a

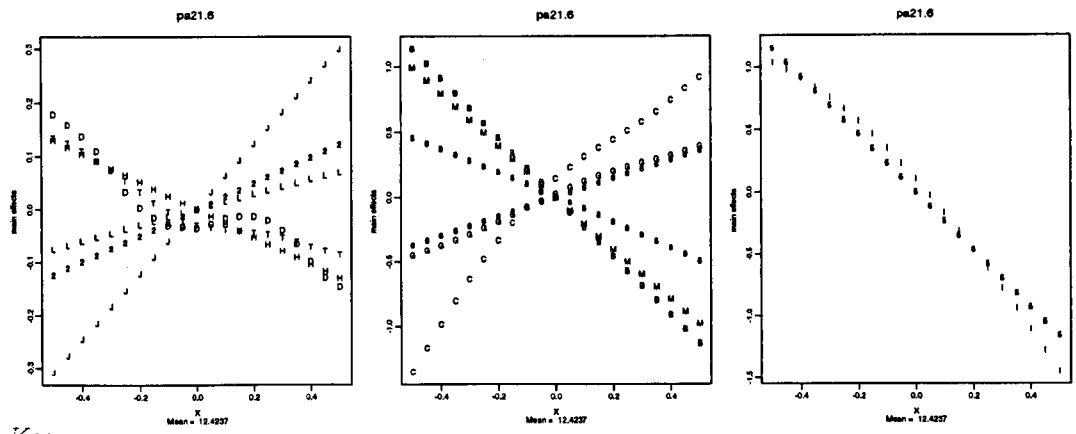


Figure 5.8: Main effects plot for block 1 with response at 200Hz

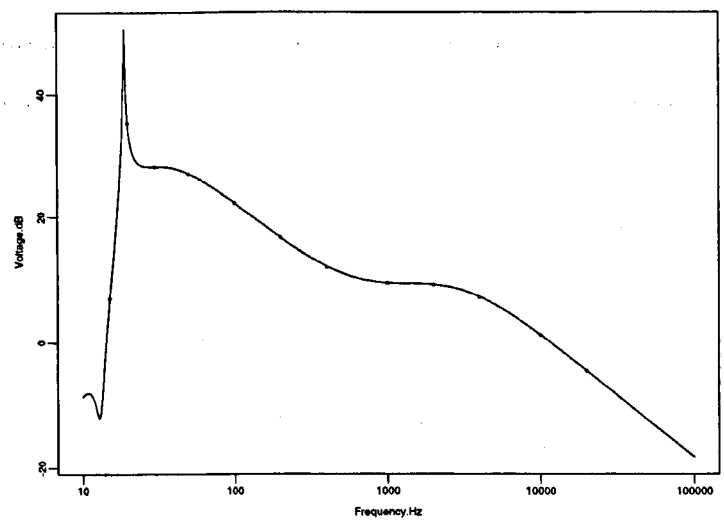


Figure 5.9: PA20 circuit - voltage output and measuring points

	Block1	Block2	Block3	Total	Full Circuit
Variables	24+5	35+5	19	88	78
Runs	60	82	48	190	166

Table 5.5: Experiment statistics

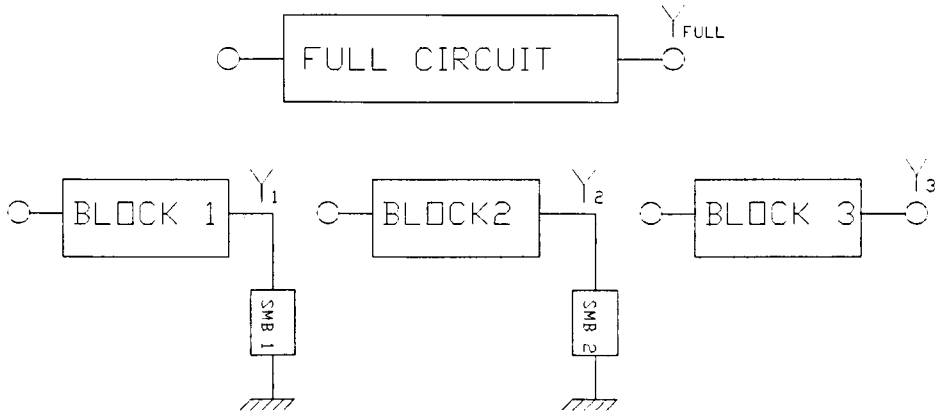


Figure 5.10: Partitioning of PA20 for analysis

methodology is described in Chapter 6, however to demonstrate the partitioning and modelling aspects of the analysis three responses corresponding to frequencies of 20Hz, 200Hz and 10KHz were chosen.

5.5.2 Transient analysis

To analyse a system partitioned into blocks in this way, simulation needs to proceed from the first block in the scheme, as follows.

i. Perform an AC analysis on each sub-circuit whose load is to be modelled to determine $\mathcal{Z}in_{nom}$ (input factors set to nominal).

ii. Determine values for R_L and C_L, the nominal load parameters, at the frequency of simulation.

iii. Add R_L and C_L to the end of the block under experimentation (Figure 5.6).

iv. Perform an experiment on the first block keeping R_L and C_L at their calculated nominal values.

v. Measure Y_1, the output of the first block, for each trial and from the range of values obtained calculate the mean value and associated tolerance for this range. Use this information as the input signal to the next block.

vi. Repeat from Step 2 until all blocks have been analysed.

vii. Build models (Section 5.6.1).

5.5.3 Variable load blocks

Taking the value of $\mathcal{Z}in$ at the nominal circuit level assumes that variations in the load block do not influence circuit response. There is a difference in approach for the two types of experiment. Because the Transient Analysis needs the input signal to be explicitly defined and fed in to each block the experiments need to be done working through the blocks starting from the first block. The $\mathcal{Z}$in for the next block thus needs to be worked out first using one extra simulation. The nominal value of $\mathcal{Z}$in is therefore taken to minimize the cost of performing extra simulations.

For the AC analysis the input signal is kept constant since we are dealing with transfer functions of the circuit. This allows a 'last block first' approach to collect data on $\mathcal{Z}$in for the block behind while doing the experiment on the block infront. A spread of SMB parameters can therefore be used during the experiment to follow more closely the changes in $\mathcal{Z}$in encountered with different parameter settings for the block infront. This spread of parameters is used to build the block model and then replaced by a 'typical' set of parameters for prediction.

5.6 Model building and verification

The objective of analysing a circuit is the optimization of the circuit design in some sense. By modelling the behaviour of the circuit and constructing an emulator of the circuit simulator which is less expensive to evaluate the design becomes easier to optimize. The response of a simulated circuit is described using a model with the independent variables being the circuit inputs (parameter values, signal inputs etc.) and the dependent variables being the circuit outputs (frequency response, amplitude etc.). The statistical model used to emulate the circuit simulator is fully described in [10] and used in [1] to optimize the design of two IC circuits. It is computed from data obtained by conducting a computer experiment. That is the circuit is simulated according to an experimental design plan and the circuit responses measured. A description of the model can be found in Chapter 2.

5.6.1 Analysis of decomposed circuits

The circuit is partitioned into sub-circuits and a suitable sub-emulator is found in turn for each sub-circuit. The sub-emulators are combined to create an emulator of the behaviour of the whole circuit. Construction of the emulator depends on which type of simulation is used and on the treatment of the load block parameters L_p. The order of simulation of sub-circuits and hence construction of the sub-emulators becomes important when considering the load blocks since L_p for one sub-circuit is obtained from the neighbouring sub-circuit. A schematic of the combination of separate block models is described for the generalised two block system in Figure 5.11.

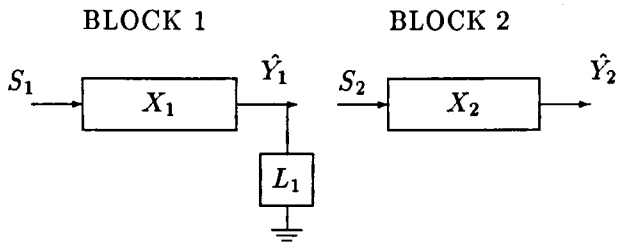


Figure 5.11: Model Schematic

Model building with transient analysis

Following the procedure outlined in Section 5.5.2 we obtain models for each block of the system. For the case of a system divided in two (see Figure 5.11) these are of the following form:

$$\hat{Y}_1 = f_1(S_1, X_1) \qquad \hat{Y}_2 = f_2(S_2, X_2) \tag{5.10}$$

where S_1, S_2 are signal inputs and X_1, X_2 are parameter settings for blocks 1 and 2 respectively. Note that the load term L_1 does not appear in the model if it is fixed at a nominal level for the experiments (see Section 5.4.2). The output from block 1 is the input to block 2 (step v. Section 5.5.2) thus

$$S_2 = \hat{Y}_1 \tag{5.11}$$

which gives

$$\hat{Y}_2 = f_2(f_1(S_1, X_1), X_2) \tag{5.12}$$

Model building with AC analysis

Building a full system model from block models using AC analysis is different from transient analysis case because for simulation in the frequency domain the output of one block is not fed to the next. In AC analysis we are interested in the transfer function of the circuit, that is the magnitude and phase of the output signal *relative* to the input signal. By measuring the magnitude response in decibels we can add transfer functions of sub-circuits to obtain full circuit transfer functions (Equation 5.15). For the example circuit the p load block parameters, represented by the vector L_p, are included in the sub-emulators to test their importance. Examination of the sub-emulators determines the importance of L_p to the response. As L_p is shown not to be significant in the sub-emulators it is substituted with a vector of nominal parameter values L_{nom}. From the procedure in Section 5.4.1 the system of Figure 5.11 yields models of the form:

$$\hat{Y_1} = f_1(S_1, X_1, L_1) \qquad \hat{Y_2} = f_2(S_2, X_2) \tag{5.13}$$

Note that the load term L_1 is contained in the model for $\hat{Y_1}$. It should also be noted that it is a function of (S_2, X_2). To fit a full system model therefore requires the elimination of this term. One approach is to fit a separate model for L_1 of the form

$$\hat{L_1} = f_3(S_2, X_2) \tag{5.14}$$

and use this in conjunction with the first two models to build the full model. However for the example, simply fixing L_1 at its nominal level produces good results. The full

system model then becomes

$$\hat{Y}_{full} = f_1(S_1, X_1, L_{1_{nom}}) + f_2(S_2, X_2) \tag{5.15}$$

5.6.2 AC results

The methodology was applied to the preamplifier circuit (PA20) shown in Figure 5.3. An emulator of the circuit was built from three separate sub-emulators of sub-circuits formed according to the partitioning results of Section 5.3.2. The emulator was then tested against simulations of the full circuit and compared with an emulator of the full circuit built directly from simulations of the full circuit. Table 5.6 shows the results of prediction at 50 new points (the simulator $Y = f(X)$ evaluated at different parameter settings, $X_1, \ldots, X_50$) for the three responses chosen. The Root Mean Squared Error (RMSE) is defined as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(Y_n - \hat{Y}_n)^2}{n}} \tag{5.16}$$

where Y is the vector of n simulator results and $\hat{Y}$ is the vector of n predictions using the model in question (in this case $n = 50$).

$$\text{Range} = \max(response) - \min(response) \tag{5.17}$$

$\frac{RMSE}{\text{Range}}$ is therefore a measure of fractional error due to model inaccuracy as it shows the absolute error of prediction scaled by the range of prediction.

The results (Table 5.6) show the effectiveness of the partitioning technique in emulating a large circuit. The emulator built from sub-emulators is at least as accurate as the full

Response		Block1	Block2	Block3	Total Block	Full Circuit
20Hz	RMSE	3.42	4.21	1.76e-1	5.32	6.11
	Range	15.5	15.2	25.9	55.65	55.65
	$\frac{RMSE}{\text{Range}}$	2.20e-1	2.77e-1	6.8e-3	1.46e-1	1.68e-1
200Hz	RMSE	4.23e-1	5.25e-2	2.67e-1	4.15e-1	4.05e-1
	Range	7.24	5.1	10.3	15.76	15.76
	$\frac{RMSE}{\text{Range}}$	5.84e-2	1.03e-2	2.59e-2	2.63e-2	2.57e-2
2000Hz	RMSE	2.78e-1	2.50e-2	2.84e-1	4.75e-1	4.83e-1
	Range	7.59	4.9	22.87	24.8	24.8
	$\frac{RMSE}{\text{Range}}$	3.66e-2	5.1e-3	1.24e-2	1.8e-2	1.95e-2

Table 5.6: Model results

circuit emulator and more accurate in the first and third cases. The general ability to model the first point accurately is hampered by the steepness of the response curve at that point (point 2 in Figure 5.9).

5.7 Discussion

The methods described partition circuits in a linear fashion for more efficient analysis. Decomposition of more complex systems may produce blocks which are connected to each other in more complex ways. In these cases the method of experimentation may need to be more sophisticated and further work is needed in this area to provide a more generalised methodology of experimentation.

In the case study models were built to predict circuit response at three specific points along the response curve. Table 5.6 shows that the models for the circuit response at 200Hz and 2000Hz are very accurate and that the models constructed from blocks for the points examined are at least as accurate as the models built for the full system. Both full and block models built for the response at 20Hz however are not as accurate due to

the difficulty of modelling the response at such a steep point along the response curve (Figure 5.9). A more comprehensive model of the system could be obtained by modelling the whole response curve as described in Chapter 6.

The results show the technique of decomposing a circuit into sub-circuits for analysis compares favourably with analysis of the whole circuit, producing slightly better results. The ability to analyse large systems in blocks means that systems previously too large for analysis can be tackled using the methodology outlined here.

5.8 Related work

Dividing complex circuits into sub-circuits for analysis has its foundations in graph decomposition techniques. Analysis of decomposed circuits tends to be more efficient because it takes advantage of the latency inherent in analysing large circuits [3]. It may also be the case that in a given system, a sub-circuit is repeated a number of times allowing the same emulator model to be used thereby increasing the utility of the approach. Representing circuits with graphs highlights the topological relationship between components [2]. Graphs of systems can be decomposed both directly [12] and ierarchically [13] to increase efficiency and are also used to solve the VLSI min-cut problem [9, 14], see Section 2.7. Topological analysis of circuits is also being used in transistor circuit analysis [11].

5.9 Conclusion

In this chapter a method has been presented to decompose electronic circuit designs, experiment on the resultant blocks and produce emulator models of the blocks to

emulate the full circuit. The resulting model can be used to optimize the circuit in the manner described in Chapter 4. Methods for dealing with transient as well as AC circuit analysis are presented and an example of a circuit split into three blocks for AC analysis is given. Model building results show that considerable computation time is saved using the partitioning procedure for a minimal loss in model accuracy. The model produced for the circuit can be used in an optimization procedure to complete the RED process. The method is particularly suited to modelling systems where clear boundaries can be established between sub-systems. Further work should allow the generalisation of these methods to a wider range of circuits for different forms of circuit partition.

References

[1] Maria C. Bernardo, Robert Buck, Lishin Liu, William A Nazaret, Jerome Sacks, and William J Welch. Integrated circuit design optimization using a sequential strategy. *IEEE Trans. Comp. Aided Des.*, CAD-11:361–372, 1992.

[2] L K Chen, B S Ting, and A Sangiovanni-Vincentelli. An edge-oriented adjacency list for undirected graphs. *Int. J Circ. Theory & Appl.*, 7:55–63, 1979.

[3] An-Cheng Deng. On network partitioning algorithm of large-scale cmos circuits. *IEEE Trans. Circuits & Systs.*, 36:294–299, Feb 1989.

[4] C Desoer and E Kuh. *Basic Circuit Theory*, page 654. McGraw-Hill, New York, 1969.

[5] C M Fiduccia and R M Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181, 1982.

[6] Gary D Hachtel and Alberto L Sangiovanni-Vincentelli. A survey of third-generation simulation techniques. *Proceedings of the IEEE*, 69:1264–1280, Oct 1981.

[7] Balakrishnan Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. Computers*, C-33:438–446, May 1984.

[8] G Kron. *Diakoptics: The Piecewise Solution of Large Scale Systems.* MacDonald, London, 1963.

[9] F Luccio and M Sami. On the decomposition of networks in minimally interconnected subnetworks. *IEEE Trans. Circuit Theory*, 16(2):184–188, 1969.

[10] Jerome Sacks, William J Welch, Toby J Mitchell, and Henry P Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–435, Nov 1989.

[11] A Sarmiento Reyes. Efficient pertitioning-based method to determine the upper bound on the number of operating points in transistor circuits. *IEE Trans. Circuits and Systems*, 141(4):258–264, 1994.

[12] J Starzyk. Signal-flow-graph analysis by decomposition method. *IEE Proceedings Pt-G*, 127:81–86, Apr 1980.

[13] Janusz Starzyk and Edward Sliwa. Hierarchic decomposition method for the topological analysis of electronic networks. *Circuit Theory And Applications*, 8:407–417, 1980.

[14] L Tao and Y C Zhao. Effective heuristic algorithms for VLSI circuit partition. *IEE Proceedings-G*, 140(2):127–134, April 1993.

[15] G W Wasilkowski and H Woźniakowski. There exists a linear problem with infinite combinatory complexity. *Journal of Complexity*, 9:326–337, 1993.

[16] Anatoly A. Zhigljavsky. *Theory of Global Random Search*, chapter 4. Kluwer Academic Publishers, 1991.

Chapter 6

Circuit response modelling for robust design

6.1 Introduction

The use of circuit simulators such as SPICE [3] is essential to the development of complex electronic circuits where they are often used as a fundamental part of an analysis scheme such as Monte Carlo or Robust Circuit Design as in Chapter 3. In circuit simulation one seeks to measure the response of a circuit to a given input. This is generally represented as voltage or current plotted against time or frequency. We are interested in the frequency analysis of circuits where the designer specifies the range and resolution of frequency values for simulation and the circuit response is measured at these particular values. The frequency response of the circuit is typically displayed by plotting these values and connecting them with straight lines using simple piecewise linear interpolation. To obtain particular response values, such as peak frequency, from

the curve the computer will return the value of the nearest frequency point. As there is only linear interpolation between these values the accuracy of the result is dependent on the density of frequency points. The output of the circuit is a function of frequency $Y(\omega)$. As highlighted in Chapter 5 when faced with a response curve a typical solution is to model individual points on the curve to obtain responses which are scalar. In this Chapter the problem of modelling a response *function* is addressed.

Section 2.3.2 provides a short description of the DACE model, Section 6.2 explores the problem of modelling a response function while Section 6.3 describes a strategy for modelling responses and improving the computational efficiency of the simulator during RED experiments. Section 6.4 follows with an example.

6.2 Modelling a response function

One way of modelling a response function $Y(\omega)$ is to include ω in the vector $\mathbf{x} = x_1, \ldots, x_n$ of n input factors for an RED experiment. Instead of the usual arrangement where we model

$$Y(\omega) = f(\mathbf{x}) \tag{6.1}$$

with

$$\hat{Y}(\omega) = \hat{\mathcal{F}}(\mathbf{x}) \tag{6.2}$$

at discrete values of ω, we can use the model

$$\hat{Y}(\omega) = \hat{\mathcal{F}}(\mathbf{x}, \omega) \tag{6.3}$$

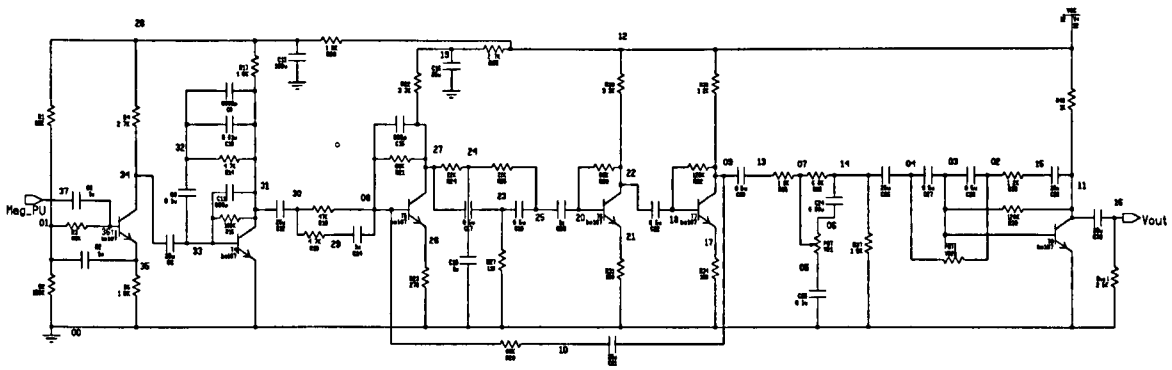


Figure 6.1: PA20 circuit

to model the entire response function in the range of ω used by the simulator for the experiment.

6.2.1 Simulation

The circuit described in Chapter 5 is also used in this chapter to explore the ideas presented. the study concentrates on the magnetic pickup transducer input ('Mag_PU' in Figure 6.1) for simplicity. Overall there are 79 input factors for the experiment, $n = 78$ for the vector $\mathbf{x}$ of circuit parameters plus one for the frequency factor. The circuit is simulated in the frequency domain in the log scale over the range 15Hz to 25KHz with 100 points/decade, this gives a response curve with $m = 324$ simulation points, that is m evaluations of equation 6.1 at different ω values. The curve is given in Figure 6.7. The $n = 78$ usual input factors gives an RED experiment of 166 simulations using a Latin hypercube sampling design of size $2n + 10$ (see Section 3.3.2).

6.2.2 Model building

In order to construct a model the factor ω needs to be included as a factor with the input vector $\mathbf{x}$. Following the procedure above gives $2n + 10 = 166$ frequency response curves each at m different frequency values which correspond to 166 different configurations of $\mathbf{x}$ according to the design plan. This data needs to be rearranged prior to model building so that the response is a single number and ω is included with $\mathbf{x}$.
For each trial in the experimental design plan $\mathbf{s} = \mathbf{x}_1, \ldots, \mathbf{x}_{2n+10}$ there are m response values corresponding to the m values of ω. That is to say the effective number of trials for the experiment becomes $m \times (2n + 10) = 53784$ for our example. To include all the data generated by the experiment yields an input matrix of $(n + 1) \times (m \times (2n + 10))$ which for our example generates a 79×53784 matrix. The model building exercise for such an experiment is extremely large requiring roughly 30Mb of storage space for the experiment data, this prohibits model building which requires even more memory. For the purposes of demonstration the factor ω is treated as an ordinary factor of the Latin hypercube sampling design plan and is divided into $(2n + 10)$ evenly spaced values over the range 15Hz to 25Khz (log scale) and included with the n input factors, preserving the number of trials at $(2n + 10)$.
In practice the original LHS design plan is used with a single column added for ω with the values of this column set at $(2n + 10)$ evenly spaced values. The corresponding response values are then single points from each of the $(2n + 10)$ curves generated in the RED experiment.

6.2.3 Results

```
MLE RESULTS

NUMBER OF LINEAR MODEL PARAMETERS IS: 1
Variable                  Beta        Std. Err.     t-val
-------------------------------------
Constant                6.3213e+00   0.0000e+00          Inf

GAMMA=   0.0000
THETA=  2.6753e-02 2.2063e-11 1.4597e-07 1.6416e-03 4.7643e-09
THETA=  1.8669e-08 9.0892e-07 3.6376e-11 1.3547e-01 3.3688e-02
THETA=  4.7643e-09 6.7353e-03 1.3382e-11 3.1131e-08 5.1425e-05
THETA=  4.7643e-09 4.9230e-12 1.3532e-02 5.4729e-09 5.7643e-09
THETA=  5.7643e-09 1.0008e-01 5.7296e-03 1.6669e-08 9.4529e-07
THETA=  4.7643e-09 4.7643e-09 1.2608e-01 8.0791e-03 9.5453e-02
THETA=  3.8609e-05 4.7643e-09 1.2066e-01 1.0882e-01 6.7353e-03
THETA=  4.7643e-09 1.7676e-02 5.2969e-03 9.6694e-08 4.9230e-12
THETA=  4.7643e-09 1.6669e-08 4.7643e-09 4.7643e-09 5.7643e-09
THETA=  4.7643e-09 5.7643e-09 4.7643e-09 3.6500e-07 7.7643e-09
THETA=  4.7643e-09 1.4517e-02 5.3585e-03 9.6694e-08 4.7643e-09
THETA=  7.6321e-03 1.0101e-11 4.7643e-09 4.7643e-09 5.9526e-05
THETA=  3.1719e-06 4.7643e-09 1.6669e-08 3.3500e-07 4.7643e-09
THETA=  5.7643e-09 1.4202e-05 2.0723e-07 2.3824e-08 1.9241e-06
THETA=  2.1623e-02 7.0809e-07 1.2387e-07 6.4419e-03 1.1810e-11
THETA=  5.7643e-09 2.1964e-02 2.8077e-03 2.1417e+01
POWER=  1.7797e+00 1.6499e+00 1.9994e+00 1.9999e+00 1.7898e+00
POWER=  1.9984e+00 1.9994e+00 1.0946e+00 2.0000e+00 1.0095e+00
POWER=  1.7898e+00 1.4950e+00 1.6635e+00 1.8995e+00 1.5590e+00
POWER=  1.7898e+00 1.6635e+00 1.0018e+00 1.9860e+00 1.9994e+00
POWER=  1.9994e+00 1.9842e+00 1.9890e+00 1.9994e+00 1.9994e+00
POWER=  1.7898e+00 1.7898e+00 1.9999e+00 1.9890e+00 1.9997e+00
POWER=  1.9626e+00 1.7898e+00 1.9977e+00 1.9991e+00 1.0038e+00
POWER=  1.7898e+00 1.9991e+00 1.9626e+00 1.9560e+00 1.7633e+00
POWER=  1.7898e+00 1.9994e+00 1.7898e+00 1.7898e+00 1.9994e+00
POWER=  1.7898e+00 1.9994e+00 1.0009e+00 1.9994e+00 1.9984e+00
POWER=  1.7898e+00 1.9954e+00 1.9780e+00 1.9560e+00 1.7898e+00
POWER=  1.7898e+00 1.7818e+00 1.7898e+00 1.7898e+00 1.9994e+00
POWER=  1.9862e+00 1.7898e+00 1.9994e+00 1.9925e+00 1.7898e+00
POWER=  1.9994e+00 1.7220e+00 1.9984e+00 1.9984e+00 1.9985e+00
POWER=  1.9968e+00 1.9925e+00 1.9925e+00 1.9455e+00 1.9175e+00
POWER=  1.9994e+00 1.4105e+00 1.7498e+00 1.7780e+00
```

Figure 6.2: DACE model parameters for response function

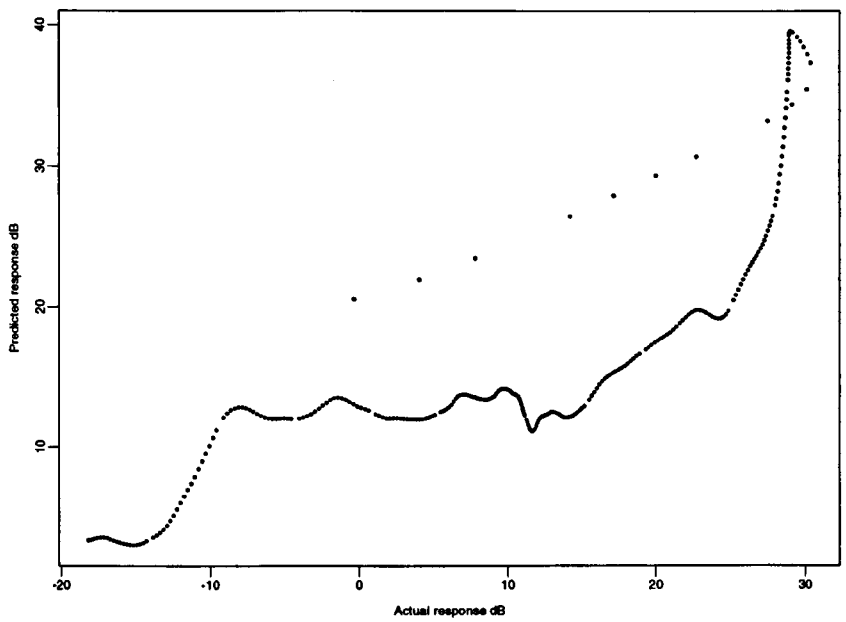


Figure 6.3: Y vs. $\hat{Y}$ for the DACE model including ω

The DACE model built from these data was used to predict the response curves generated by the original RED experiment. The ERMSE of prediction (see Chapter 3) for the model is 10.27 showing the model is not predicting accurately enough compared with models constructed without ω as an input factor (see next section). Analysing the DACE model parameters of Table 6.2 shows that the 79th factor, ω, has a value for θ 170 times larger than the next largest value. This implies that ω completely dominates the model and overshadows any effect other factors might have. This can be seen clearly in Figure 6.3, a graph of true versus predicted response values. Figure 6.4 exposes this by plotting Y and $\hat{Y}$ against ω which shows the poor prediction property of the model and both responses dependence on ω.

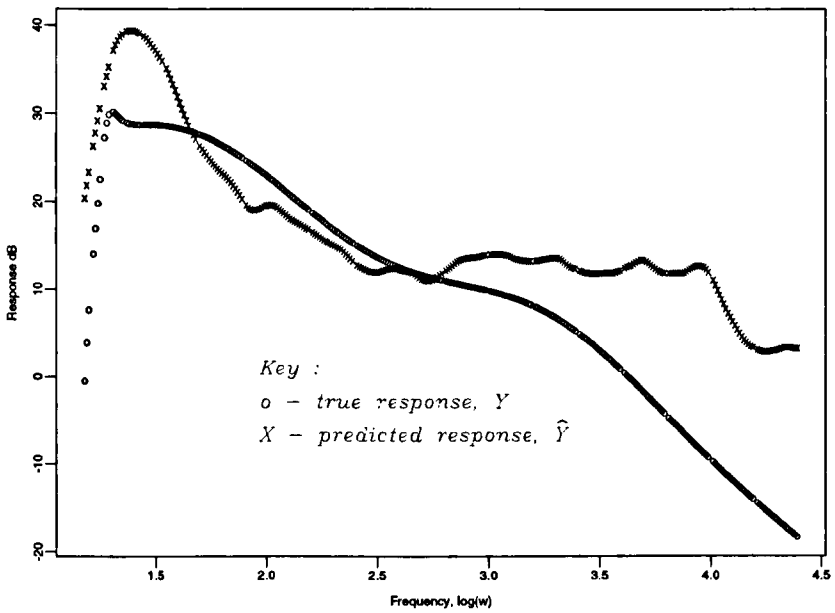


Figure 6.4: Y and $\hat{Y}$ vs. ω

6.2.4 Conclusion

The results of the model-building exercise show that the importance of ω to the function cannot be easily represented with this naive use of the DACE model. The loss of information due to the need to reduce the data set generated by the RED experiment makes the prediction model inaccurate and the dominating effect ω as an input factor tends to swamp the effects of the other input factors.

6.3 Modelling a family of functions

An alternative way of modelling the response curve to that given in the previous section is to build what will be termed a *meta-model* emulator describing the response as a function of ω from an initial simulation. This model can then be used as a controller of

the more usual response emulators at carefully selected values of ω to predict the response at any frequency point.

A necessary part of the circuit design process is the ability to simulate the circuit under different conditions (signal input, parameter values, temperature etc.) to observe changes in response either manually or as part of a design strategy. These strategies necessarily require many simulations of the same circuit at different input values producing what will be called in this Chapter a 'family' of response curves which, although unique in terms of the input values applied, display similar characteristics over a range of input values.

Once a meta-model is defined for the response of a particular circuit it can be used in conjunction with further simulations of the circuit to reproduce the response curve. The benefits of this method are twofold. First, the simulation of the circuit will be faster and computationally more efficient due to the reduced number of frequency points needed to be calculated by the computer to estimate the response. Second, the model can be used as an interpolator to estimate the value of the response between frequency points.

The novelty of this technique is that the frequency points are chosen adaptively from the first nominal or 'base' simulation. In statistical terminology the method is a two-stage adaptive sampling strategy. At the first stage the choice of the special frequency points for later use is made using a proven statistical technique, namely cross-validation (CV) [1]. Each frequency point is assessed according to the change in the accuracy of the fit as measured by the increase in root mean squared error (RMSE).

The base frequency vector for the first-stage simulation is written ω_0 the subsequent simulations are carried out for siblings with frequencies ω_1. The first stage simulation has sample size (that is the number of frequency points) equal to n_0. The siblings each

have sample size n_1 giving a total sample size of

$$N = n_0 + kn_1 \tag{6.4}$$

where k is the number of siblings.

Typically $n_1 \ll n_0$ and hence $N \ll (k+1)n_0$ which might pertain when each curve is treated equally. Here n_1 is selected at different sample sizes (see the example) to investigate the relationship between sample size and model accuracy. The key point is that the n_1 points for the siblings simulations are a fixed subset of the base sample of n_0 points for the base simulation. The procedure is summarised as follows:

Stage 1 Simulate at n_0 points then reduce sample size to n_1 using CV. This ranks each frequency point in the set ω_0 according to its importance to the model through the equation

$$\delta_i = RMSE_i - RMSE \qquad i = 1, \ldots, n_0 \tag{6.5}$$

where $RMSE_i$ is the root mean square of the model fit when the ith point is left out of set ω_0.

By taking a subset of the most important points for the model $\hat{g}(\omega_0)$, that is points with δ_i greater than a predetermined cut-off value, the sample size can be reduced to n_1 and the data fitted by a new model $\hat{g}(\omega_1)$. In practice, because of the high density of points for a large n_0, δ_i will not vary much from point to point making the choice of cut-off value difficult. A heuristic procedure to reduce the sample set by taking a combination of the most important points (highest δ_i's) and every second point from the remainder (or every fourth if there is evidence of

over-sampling) is used to compile n_1 points for re-modelling of the curve . The cycle is then repeated with n_1 being reduced further until a suitable trade-off between the sample size, n_1, and the model accuracy (RMSE) is achieved.

Stage 2 Perform all future simulations at the selected points, ω_1, and use the estimates of the model parameters (θ_i, p_i) from the first stage simulation to predict the frequency response at all intermediate points.

The predictor (see Section 2.3.2) is of the form:

$$\hat{\mathbf{g}}_s(\omega) = \hat{\beta} + \mathbf{r}_0'(\omega_1)\mathbf{R}_0^{-1}(\mathbf{g}_s(\omega_1) - \hat{\beta}_0\mathbf{l}) \tag{6.6}$$

where $\mathbf{g}_s(\omega_1)$ is the set of n_1 responses from the current simulation at the frequencies in ω_1 and $\mathbf{r}_0'$ and $\mathbf{R}_0$ are taken from the modelling of the 'base' simulation curve using the subset of n_1 frequency points.

6.4 Example

Data is collected from an experiment in which an electronic circuit is repeatedly simulated at various input settings as part of a Robust Engineering Design (RED) experiment and the curve modelling procedure applied. The circuit used is an audio preamplifier with a frequency response of interest in the range of 15Hz to 20KHz. The circuit contains 78 parameters which, using a Latin Hypercube Sampling (LHS) design plan [2], were varied over 166 trials for the RED experiment. This produces a family of 166 frequency response curves showing the variation in response with respect to circuit parameter values.

Pass	Start points	Top points from CV ranking	Points 'thinned out' from remainder	Total points	Mean ERMSE
1	324	52	68 from 272	120	6.0e-04
2	120	40	40 from 80	80	1.5e-03
3	80	20	30 from 60	50	3.8e-02
4	50	20	15 from 30	35	1.4e-01
5	35	15	10 from 20	25	2.2e-01
6	25	15	5 from 10	20	4.3e-01
7	20	10	5 from 10	15	7.2e-01
8	15	5	5 from 10	10	1.7e+00

Table 6.1: Stages in sample point reduction

The circuit is simulated in the frequency domain in the log scale over the range 15Hz to 25KHz with 100 points/decade, this gives a response curve with 324 simulation points for the 'base' simulation. Inspection of the CV results show 52 points with $\delta_i > 0$, these points plus 1/4 of the remaining points (choosing every *fourth* point to thin out the set speeds up the reduction process) gives $n_1 = 120$ as an initial reduction of n_0. The model of the curve is recalculated with n_1 and CV used again to reduce n_1 further. Table 6.1 shows the effect of gradually reducing n_1. The remaining 165 'sibling' curves can then be estimated using a model comprising the parameters $\hat{p}$ and $\hat{\theta}$ from the model of the base simulation at ω_1 and the n_1 frequency responses from simulating the circuit at ω_1.

To explore the relationship between estimated RMSE (ERMSE) and the sample size, n_1, the modelling procedure is repeated for all sizes of n_1 summarised in Table 6.1. The effect of reducing the sample size can be seen in Figure 6.5 where the average ERMSE of prediction for the 165 sibling curves is plotted against sample size n_1. For example the best 25 from 324 start points account for most of the root mean squared error. This plot, and that of Figure 6.6, is achieved by comparing the predicted responses with the actual simulations at the full 324 simulation points carried out for the purpose of validating the

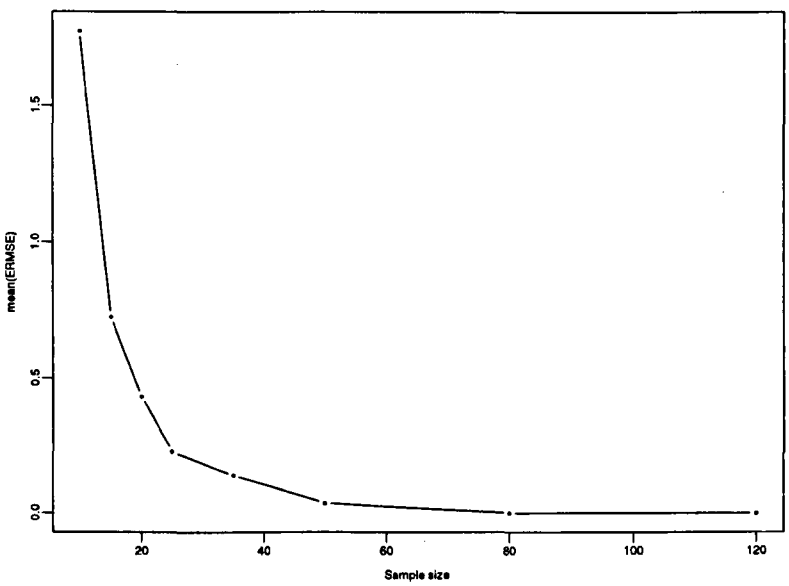


Figure 6.5: Mean ERMSE of prediction for 165 sibling curves vs. sample size n_1

method.

Figure 6.7 shows a sample distribution of the ERMSE of prediction for sibling curves at the favoured size of $n_1 = 25$.

The sample size can be reduced from 324 to 25 points without loss of accuracy (mean ERMSE for prediction of sibling curves = 0.22). This improves the efficiency of the simulation process by reducing the number of simulation points for every new circuit simulation. The chosen 25 points are shown along the circuit output curve in Figure 6.7. Referring to the prediction model, equation 2.9 in Section 2.3.2, the model fitted to the base simulation at 25 points has the parameters

$$\theta = 13.473 \qquad p = 1.4023 \tag{6.7}$$

for prediction (note that $i = 1$ for this case).

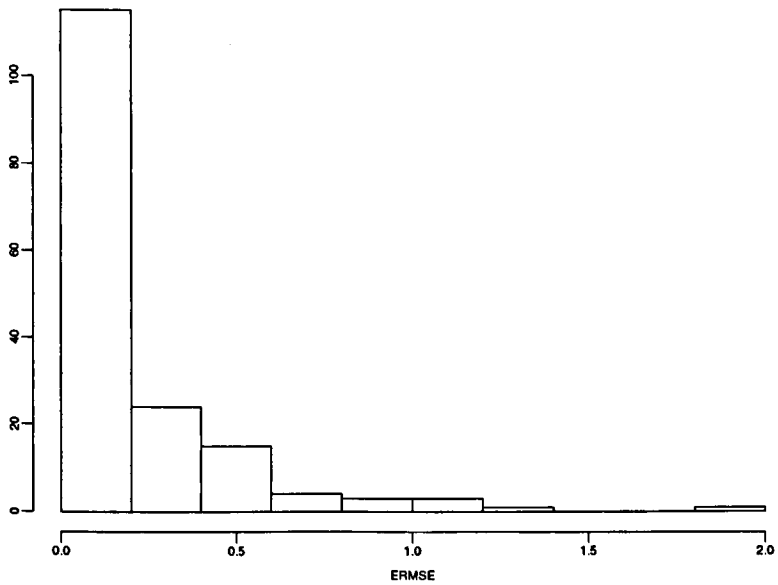


Figure 6.6: Prediction of sibling curves with model $n_1 = 25$.

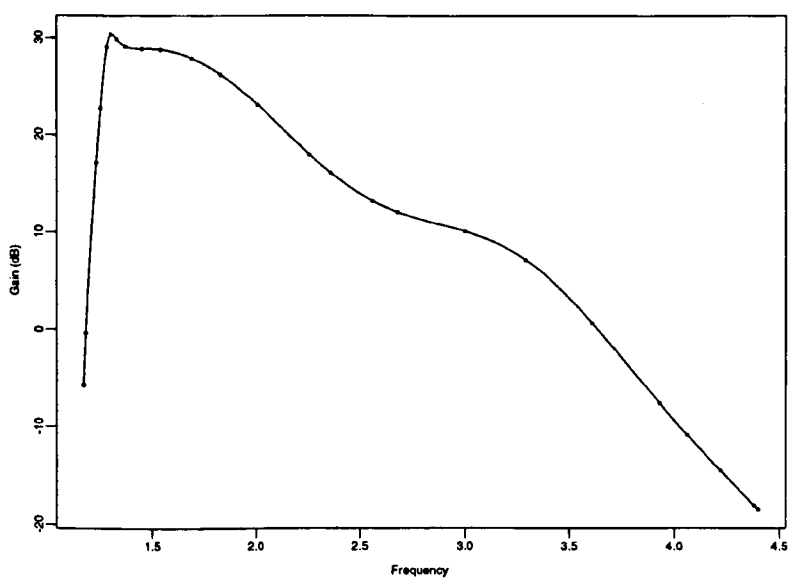


Figure 6.7: Subset of $n_1 = 25$ points taken from the base simulation.

From Figure 6.6 the sample distribution shows two curves for which the modelling process yields a relatively high ERMSE of over 2. On inspection of these curves (Figure 6.8) it can be seen that the error stems from an inaccurate initial tracking of the curve and, for the most part, the model is accurate over the remaining frequencies. If this part of the response were deemed to be critical then more points could be added here to improve the modelling.

Finally the $n_1 = 25$ point meta-model is used to predict the response curves generated by the simulator of the circuit for a new set of input observations. This involves the following procedure

i. Perform an experiment to generate a DACE model for each frequency response Y_i where $i = 1, \ldots, n_1$, generated from the selected frequency vector ω_1.

ii. For a new set of observations, predict the responses, $\hat{Y}_i$ at the n_1 frequency values.

iii. Use the predictions in the meta-model to predict the response at the other frequency values, call these $\hat{\hat{Y}}_i$.

iv. Compare the meta-model predictions with the simulator to verify the technique using the equation

$$mean\ ERMSE = \sum_{j=1}^{m} \frac{\sqrt{\frac{\sum_{i=1}^{n_0} (\hat{\hat{Y}}_{ij} - Y_{ij})^2}{n_0}}}{m} \tag{6.8}$$

The mean ERMSE of prediction at the full $n_0 = 324$ frequency points for $m = 50$ curves is 0.230 showing the technique to be accurate. Figure 6.9 shows the 324 true (simulator) response values vs. those predicted using the meta-model for the worst case. Note the

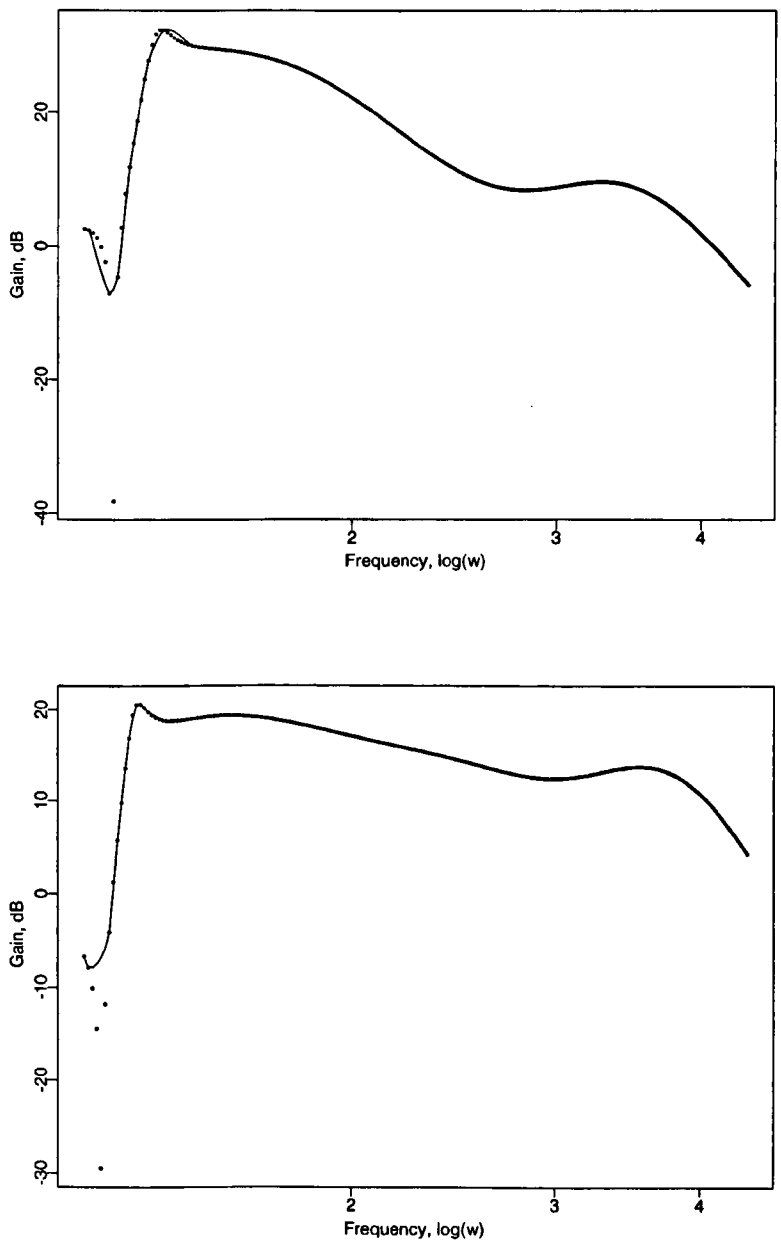


Figure 6.8: Worst two curve predictions with 25 point model.

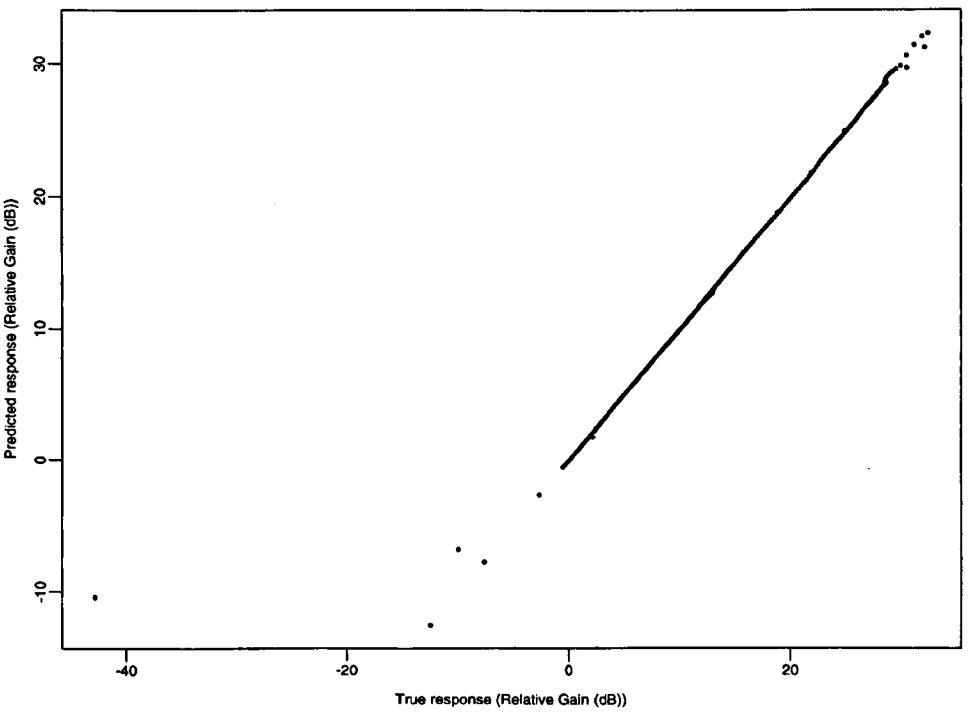


Figure 6.9: Worst predictions (ERMSE= 1.81) using meta-model.

outlier corresponding to a badly predicted point very similar to the one in the top graph of Figure 6.8.

6.5 Discussion

The principle established is that both the model parameters and the choice of frequency points can be based on a single initial simulation with substantial gain in computational time and without significant loss of accuracy. This is particularly useful in RED experiments and in circuit optimization, when large numbers of frequency curves are to be evaluated. The benefits of fast emulation of the simulator using statistical models are preserved by careful selection of a limited number of sample points.

References

[1] Noel Cressie. *Statistics for Spatial Data*, chapter 2, page 101. John Wiley and Sons, Inc., 1991.

[2] M. D. McKay, W J Conover, and R J Beckman. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21:239–245, 1979.

[3] L W Nagel. *SPICE 2. A computer program to simulate semiconductor circuits.* ERL Memo ERL-M520. Univ. California, Berkley, 1975.

[4] Jerome Sacks, William J Welch, Toby J Mitchell, and Henry P Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–435, Nov 1989.

[5] D Ylvisaker. Prediction and design. *The Annals of Statistics*, 15:1–19, 1987.

Chapter 7

Design of experiments for complex systems

7.1 Introduction

In this Chapter we explore the relationship between RED and system topology developing a method for reducing the complexity of RED for large problems. This is achieved by adding knowledge about a system to the experimental design plan of the RED experiment. Instead of partitioning a system as in Chapter 5 we experiment on the *whole* system and reduce the complexity of the experimental design plan by exploiting topological information to tear the system. The tearing procedure described in Section 2.4.2 is used and a method of constructing a blocked experimental design is presented. The methods are illustrated with a case study of the decomposition and analysis of an electronic circuit and we shall also make comparisons with experiments in which all the factors are varied at the same time.

7.2 Complexity and experimental design

In Section 1.1.3 we defined complex systems as having a large number of interacting elements and we repeat from Section 3.2.6 that the goal of RED is to model the behaviour of a system with an emulator to estimate how system input factors affect system response for optimization. Thus for a system $Y = f(X)$ with n input factors $X = x_1, \ldots, x_n$ and m responses $Y = y_1, \ldots, y_m$ we fit the emulator of Section 1.11, $\hat{Y} = \hat{f}(X)$. Experimental designs provide an efficient plan for testing a system at different combinations of X values to determine their importance on system response Y. Describing each input as a dimension, as in Section 1.2, for a range of values of X normalised to $[0, 1]$, an input space can be defined which contains all possible combinations of input factor values as $[0, 1]^n$. A point in this space corresponds with a particular vector of input factor values over the defined range. Good experimental designs minimise the number of points, or observations, required to fill the input space by spreading themselves out effectively. For complex systems the dimension of input space is large requiring a large number of points to effectively cover the input space, each point representing an evaluation of the system f. This can be prohibitively expensive. Chapter 5 was concerned with quantifying interactions between the input factors of an electronic circuit by partitioning it into sub-circuits to reduce the task of modelling. This has the effect of eliminating interactions between factors of different sub-circuits except via the *load blocks*, effectively reducing the input space for the RED experiment. With this method it is important to preserve the environment in which the sub-circuits operate through the use of load blocks emulating the surrounding sub-circuits which are assumed to be operating at nominal input factor levels. Here we avoid the use of load

blocks by considering the *whole* system for analysis but restricting the nature of the experimental design plan. If the whole system is simulated this process is equivalent to performing a single RED experiment as follows.

i. Define the sub-circuits but *do not* physically decompose the circuit.

ii. Perform an RED experiment on each sub-circuit keeping all other circuit factors at their nominal level.

iii. Build a single emulator of the simulator from the experiment.

Using circuit topology to define sub-circuits provides the opportunity to streamline the interactions between input factors which has the effect of *restricting* the input space. The experimental design plan can take advantage of this by concentrating on areas where interactions are more likely to occur. The circuit is decomposed only in the sense that the input factors are grouped together for analysis, this compares directly with the process of *tearing* a network for analysis. An important feature of the decomposition applied in this Chapter is the creation of a *border* group of variables which acts as a communication pathway between sub-circuits allowing a certain degree of interaction. This replaces and improves on the limited amount of interaction allowed via the load block parameters in Chapter 5.

7.3 Decomposition: tearing

The goal of decomposition here is to group together factors likely to share strong interactions in an attempt to reduce the input space of the experiment. The method used is related to Diakoptics [4] (ses Section 2.7.3) in that we look to decompose a

network into blocks joined by a connecting network. In Diakoptics the fact that it is much easier to invert several small matrices rather than one large one is exploited to reduce the computational effort required to solve the system equations. The connecting network serves to link the smaller matrices together. The decomposition method combines this idea with sparse matrix techniques where a matrix can be arranged in a Bordered Block Diagonal (BBD) form to tear a circuit. We use the topology of the circuit represented through the incidence matrix of its graph in the algorithm of Section 2.4.2 (an implementation of an algorithm by Zečevič and Šiljak [10]) to decompose the incidence matrix into the BBD form.

7.3.1 The incidence matrix

The first stage in decomposing a circuit involves representing the circuit with an undirected graph and using this to create an incidence matrix which is typically sparse. We repeat the process described in Section 2.4.2 to highlight the similarity between the formulation of the incidence matrix for circuit analysis and formulation for decomposition. Given a circuit represented by a graph G, where the edges are circuit components and the nodes are circuit nodes, the analysis problem can be formulated as

$$I = Y \times E \tag{7.1}$$

where I is the current vector, Y the nodal admittance matrix and E the voltage vector. This corresponds directly with circuit topology. For a circuit with n nodes and q components the nodal admittance matrix, Y, is size $n \times n$ with $2 \times q$ elements as Y is symmetric about the main diagonal.

A circuit is represented as a graph $\mathcal{G}(X, E)$ with a set of nodes $X = \{x_1, \ldots, x_m\}$ and edges $E = \{e_1, \ldots, e_q\}$ representing respectively the circuit nodes and components. This representation can be understood by comparing Figures 7.4 and 7.5. The unweighted graph is mapped into an incidence matrix in symmetric form with $2q$ elements 1 the rest 0. The first step in forming the groups of factors is to create the incidence matrix from the circuit. This is the same size as the nodal admittance matrix Y in equation 7.1, the elements of the matrix I being defined as

$$
\begin{aligned}
I_{ij} &= . , \quad \text{for nodes } i, j \text{ not connected}, \quad i, j = 1, \ldots, n. \\
I_{ij} &= X, \quad \text{for nodes } i, j \text{ connected}, \quad i \neq j, \quad i, j = 1, \ldots, n.
\end{aligned} \tag{7.2}
$$

This incidence matrix is analogous to the incidence matrix formed during the initial stages of a nodal analysis for computer-aided circuit simulation, see Chua and Lin [2].

7.3.2 Forming the blocks

The main idea is to use the block structure given by the system decomposition to aid the construction of the experimental design. Experience with this and other problems has shown that with large systems the concept of a *nominal* or centre value is important. Each individual block may affect the behaviour of other blocks, that is affect the causal link to the output. It is a mistake to wholly isolate a particular block for purposes of experimentation. An effective method is to allow an ordinary block only to 'see' the nominal levels for the other blocks. The border is treated differently. It is connected to every ordinary block and conceptually can be thought of as a communication pathway between blocks. It is allowed to have a more varied relationship in the design with each

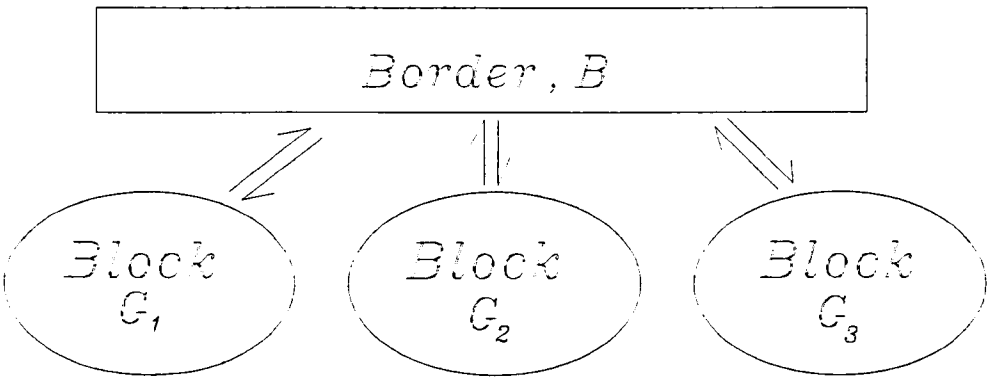


Figure 7.1: Schematic of system decomposition.

block and itself. There are two important analogies of the border in related fields which provide additional motivation:

i. in hierarchical control theory subsystems may only communicate via a 'coordinator' or 'controller' (Mesarovic, Macko and Takahara [6]), see Section 2.7.

ii. in Robust Engineering Design (specifically Taguchi methods, Section 1.3.2) the 'noise factors' may interact with any control (design) factor and the experiment is designed to allow this by crossing noise and control factors. We can think of the noise as an all-pervading medium which may potentially influence any control factor.

The schematic of Figure 7.1 shows a system decomposed, using the method described, into three blocks $G_1, \ldots, G_3$ connected by a border B.

7.4 Experimental designs

To perform computer experiments on systems with a large number of interacting factors (i.e complex systems as defined in Section 1.1.3) requires the use of design plans capable of efficiently filling the input space. Here three types of design are used in the case study which follows to model the example circuit both with and without the decomposition technique described. The designs are

i. Basic LHS designs [5].

ii. LHS due to Buck and Wynn [1].

iii. Lattice designs [8].

These design types were selected because of the size of the problem (i.e number of input factors) and the lack of knowledge about which factors and factor interactions are important for the emulator. They are preferred to the more popular Taguchi-style Orthogonal Array's (OA) for building prediction models. The benefit of orthogonality in the design plan is that it is easier to attribute response variance to input factors in studies such as ANOVA (analysis of variance). The use of full factorial design plans (OA's) is limited because the size of design plan increases rapidly with the number of factors. Fractional factorial designs preserve orthogonality and reduce the size of design plan by not searching the input space fully for interactions, preferring to concentrate on main effects. This can lead to poor prediction models if there are significant interactions between input factors. Knowledge about interactions allows full factorial designs to be reduced intelligently and can act as alternatives to the more usual space-filling designs.

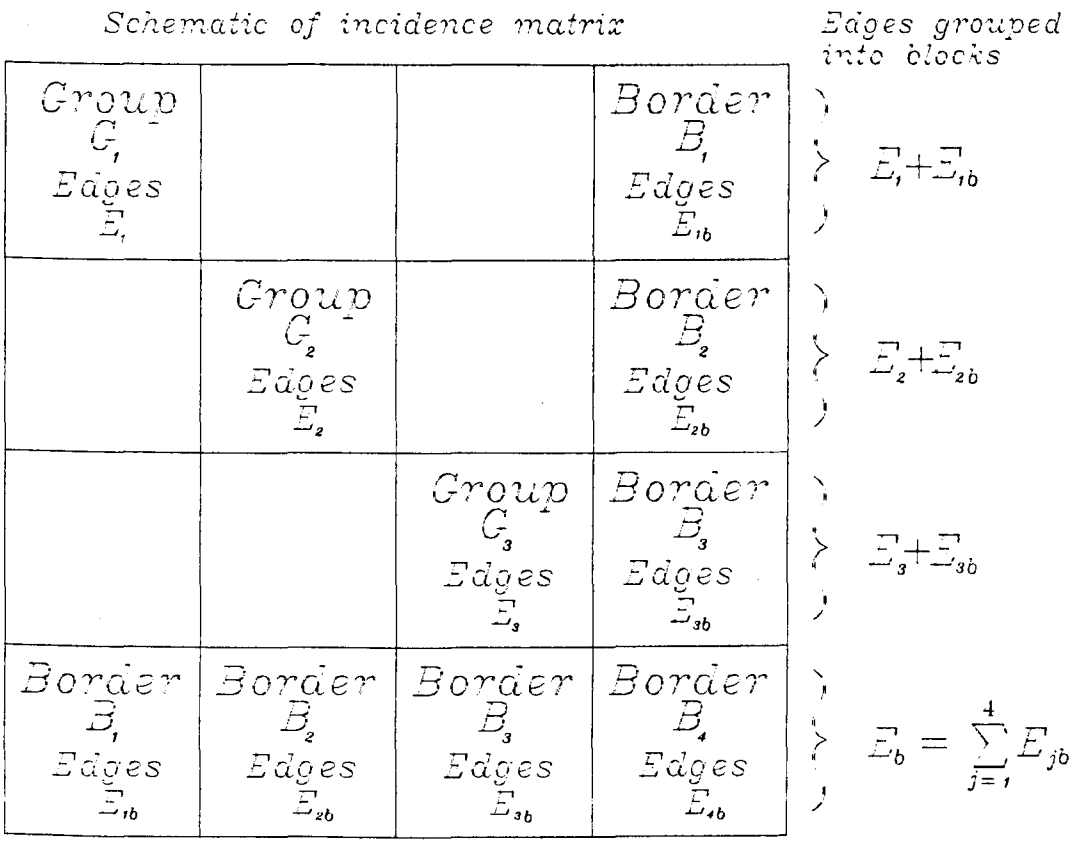


Figure 7.2: Schematic of BBD matrix.

The construction of basic and improved LHS designs and integer lattices is discussed in Section 2.3.1.

7.5 Building the experimental design plan

The experimental design plan reflects the final graph produced by the partitioning algorithm as follows. Each factor is represented by an *edge*.

i. For each block j all its edges E_j are grouped together with the set of edges in the border, E_{jb} with which it has a common node in the final graph.

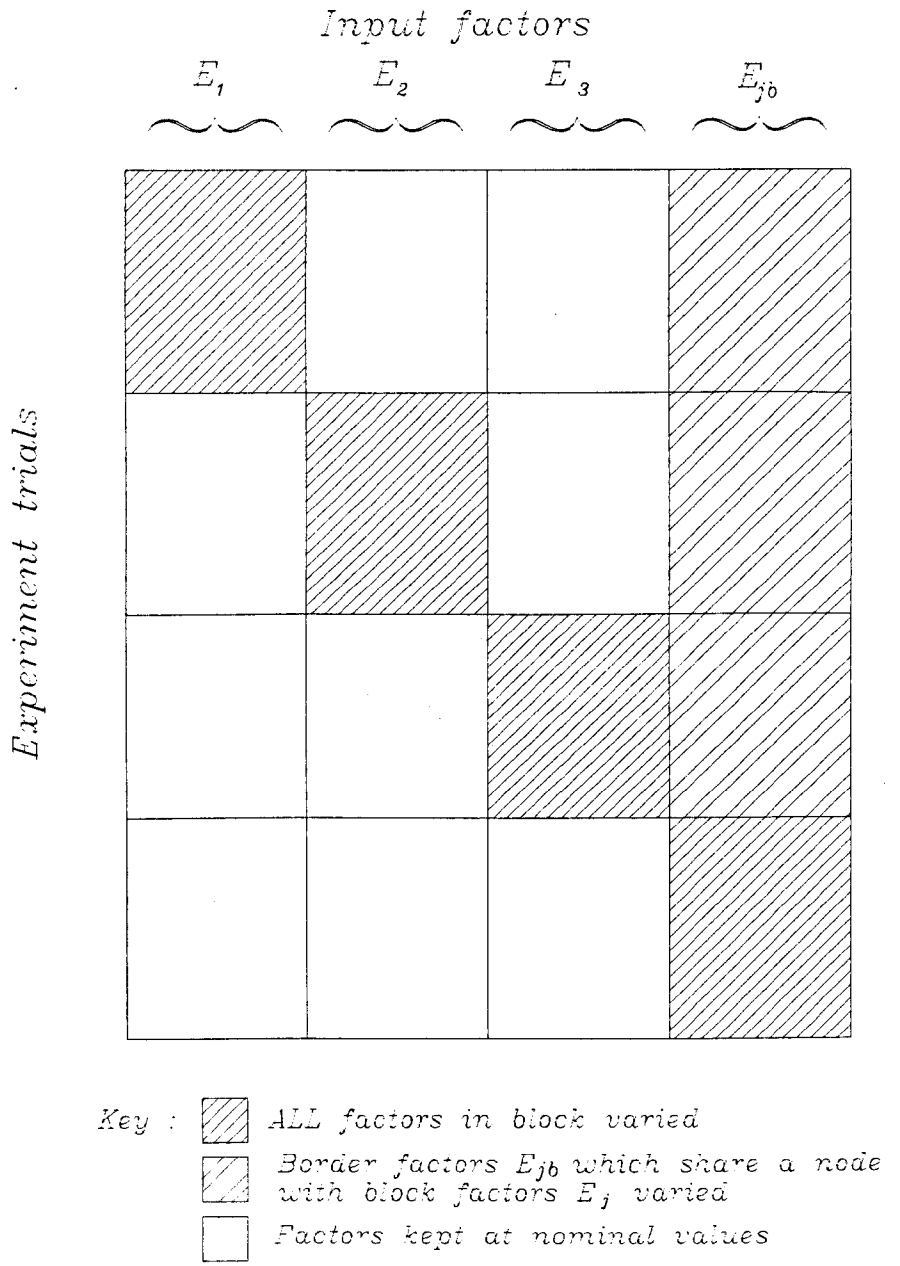


Figure 7.3: Schematic of full experimental design plan.

ii. For block j of the experiment the factors, given by $E_j \cup E_{jb}$ are varied while all other factors are set at the *nominal* value, which is here taken to be the central value of the range of the factor. This is repeated for $j = 1, \ldots, m$. This part of the experiment can be considered as a one-block-at-a-time experiment.

iii. The border factors, E_b, are varied in their own experiment in which all other factors are set to nominal.

This is illustrated in Figures 7.2 and 7.3 which show how the experimental design plan is derived from the decomposed incidence matrix. Figure 7.3 should be compared with a basic experimental design plan (see Figure 3.2 for an example) to highlight the new structure. Again, it is important not to consider the separate blocks as independent experiments. While runs are conducted varying the levels of the factors within a block the other blocks are still 'active' it is only that the complexity of the experiment is reduced using the nominal settings. In effect we are operating in a *restricted* region of input space closer to the overall nominal levels, or centre point of the whole experiment.

7.6 Case study

7.6.1 The system

An example circuit is used to develop the methods and is modelled using three different design plans. Emulators of the circuit simulator are constructed and verified over both full and restricted input spaces. The circuit is shown in Figure 7.4. It is the preamplifier circuit used in Chapter 6 and is designed to provide input to an audio amplifier and accept a wide variety of signal types. The output studied here is the frequency response

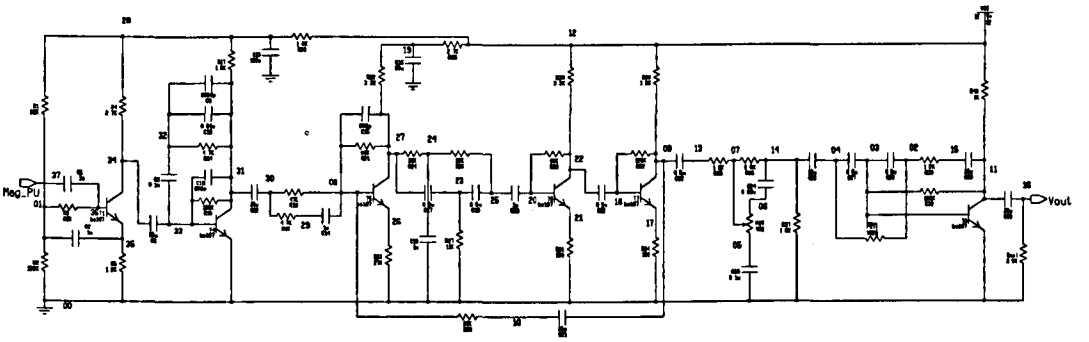


Figure 7.4: Example circuit.

presented as characteristic curves of relative amplitude in decibels (dB) against the log of frequency ($\log(\omega)$). Five points $Y_1, \ldots, Y_5$ on the response curve are chosen as responses to be modelled, corresponding to frequencies $\log(\omega) = 2.07, 2.33, 3.15, 4.40, 5.19$ respectively. In the experiments the factors represent the levels of resistance and capacitance for 60 different circuit components. The simulator AccuSim, supplied by Mentor Graphics (UK) Ltd., was used with the full integrated module for performing robust design on electronic circuits described in Chapter 3.

7.6.2 Results

For the example circuit the incidence matrix I is given in Table 7.1, note the sparsity of the matrix. This property is exploited to decompose the matrix into the BBD form of Table 7.2. The algorithm used to form the BBD matrix from the incidence matrix is fully described in Chapter 2. Figure 7.5 gives the final graph for the torn circuit. Table 7.2 shows the results of decomposition. The elements of the BBD matrix correspond with circuit components. Each block represents a group of connected components, the groups themselves connected with each other via the components in the border group. The BBD matrix is used to decompose the graph of the circuit to give the

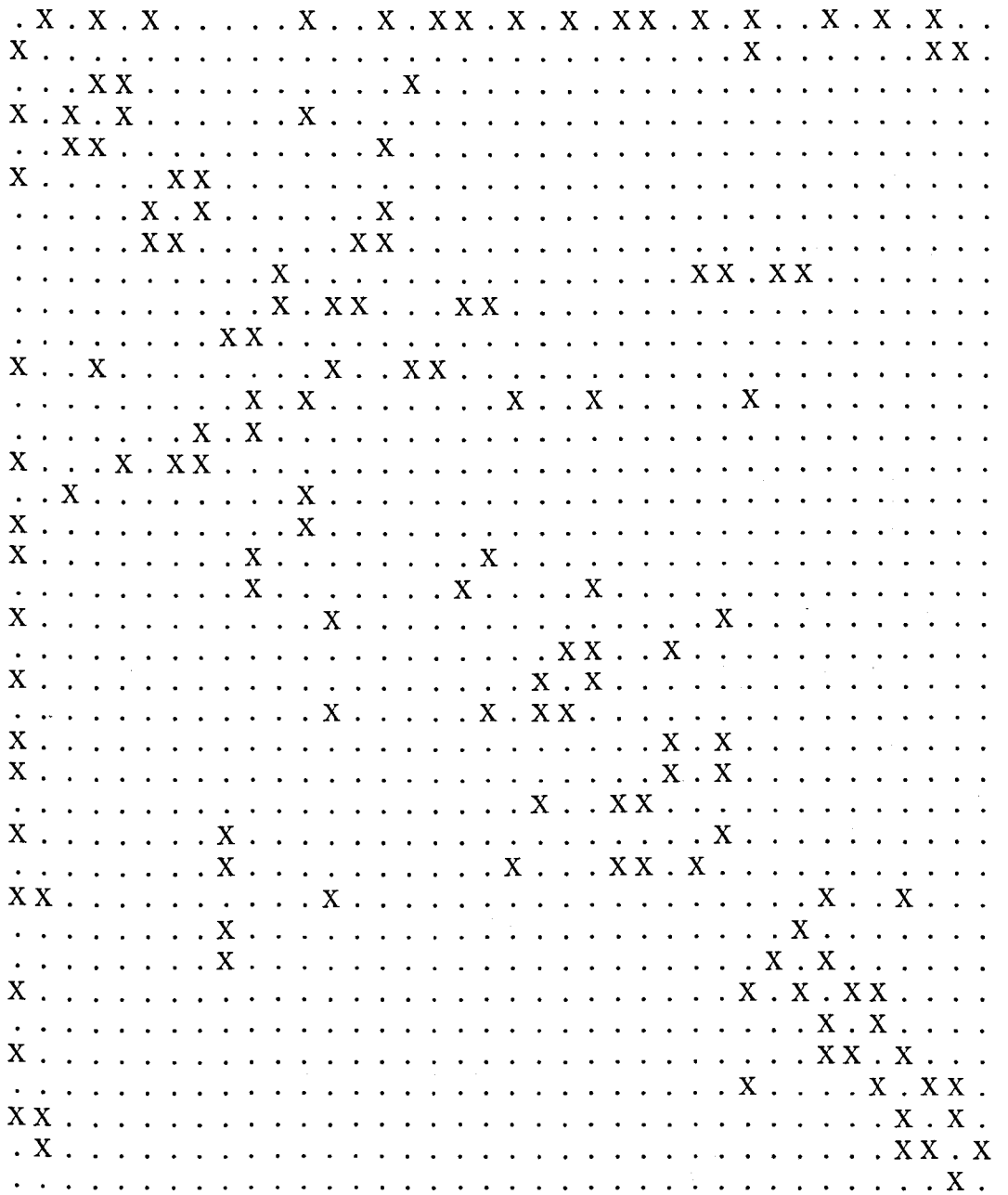


Table 7.1: Sparse incidence matrix from PA20 circuit

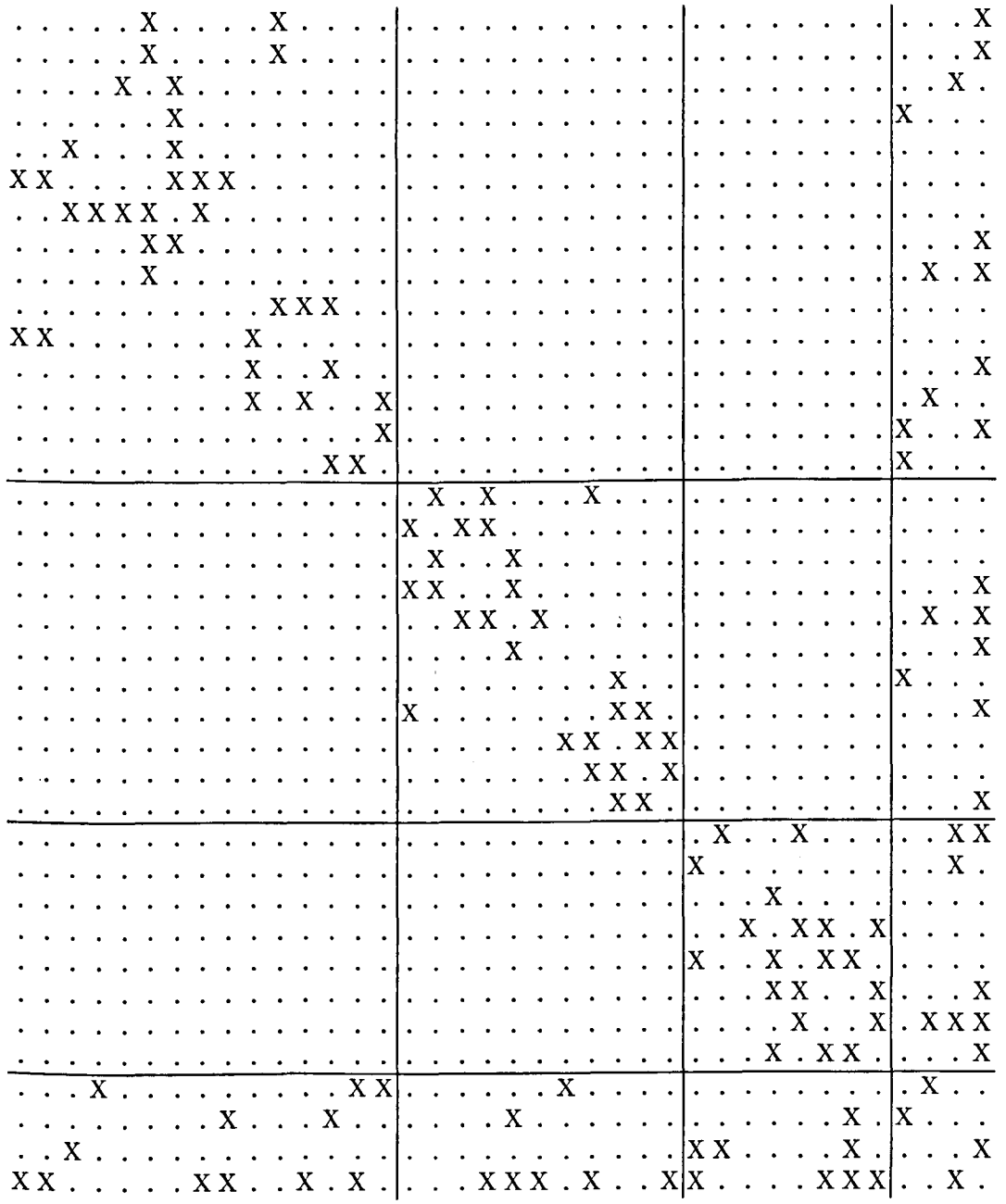
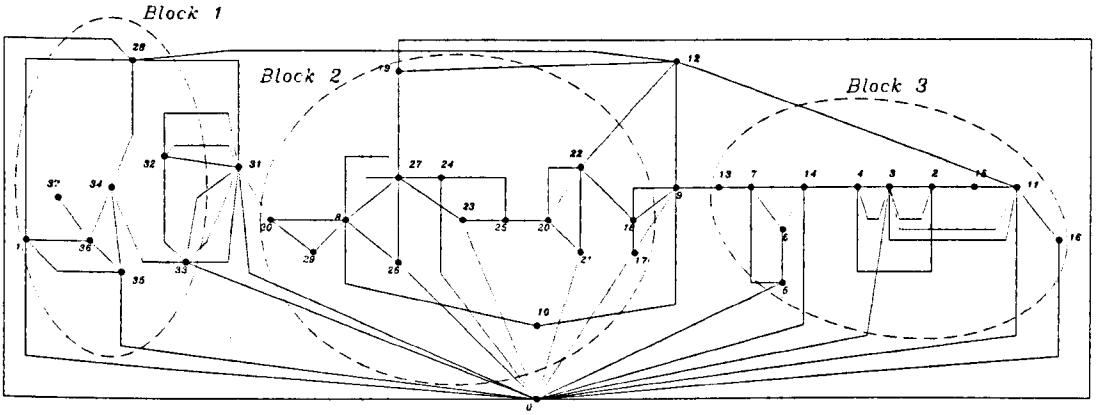


Table 7.2: Bordered block matrix from sparse matrix



Block 1 (8 nodes) : 1 28 32 33 34 35 36 37
Block 2 (11 nodes) : 2 3 4 5 6 7 11 13 14 15 16
Block 3 (15 nodes) : 8 10 17 18 19 20 21 22 23 24 25 26 27 29 30
Border (4 nodes) : 0 9 12 31

Figure 7.5: Graph of circuit.

groups highlighted in Figure 7.5. The border components are not grouped but act as the connecting network through which groups can interact, similar to the connecting network of Kron's Diakoptics.

Although the new block structure affects the experiment we have not, in this study, allowed it to affect the initial model. Thus we fit the DACE model of Section 2.3.2, fitting the parameters by maximum likelihood in the normal way. A more sophisticated approach would take into account the sparse matrix methodology in the internal numerical analysis of the statistical package itself. For example the estimation of the parameters of the covariance function may be facilitated.

The final block sizes selected by the algorithm were 25, 19 and 17 factors (edges) for ordinary blocks and 27 for the border. Table 7.3 presents the results of the experiment using the block structure and using an experiment ignoring the block structure and for three designs: D_1: simple Latin Hypercube, D_2: Improved Latin hypercube and D_3: a

Design type	Response	Full model predictions		Block model predictions	
		Full space	Restricted space	Full space	Restricted space
D_1	Y_1	3.740	2.361	5.029	2.381
	Y_2	0.432	0.316	0.610	0.324
	Y_3	0.195	0.139	0.283	0.148
	Y_4	0.179	0.116	0.270	0.128
	Y_5	0.150	0.091	0.281	0.141
D_2	Y_1	3.774	2.664	4.501	4.501
	Y_2	0.422	0.273	0.416	0.416
	Y_3	0.246	0.173	0.264	0.264
	Y_4	0.246	0.142	0.229	0.229
	Y_5	0.153	0.089	0.211	0.211
D_3	Y_1	4.378	2.467	6.397	3.563
	Y_2	0.467	0.254	0.698	0.353
	Y_3	0.202	0.106	0.433	0.223
	Y_4	0.187	0.108	0.335	0.168
	Y_5	0.170	0.089	0.349	0.171

Table 7.3: Mean ERMSE for prediction at 500 points - 60 variable model

lattice design. For the latter the primitive root method ((ii.) in Section 2.3.1) was used with prime power block sizes as close as possible to the values used in the first two experiments. Thus, the sample sizes for D_1 and D_2 for the blocks and border were respectively 60, 48, 44 and 64 and for the lattice, D_3, the primes 61, 47, 47 and 61; in both cases the total sample size is 216.

The entries of Table 7.3 are the mean squared error (MSE) of prediction at 500 test points (trials) selected independently by a simple Latin Hypercube design. For each trial there are five outputs $Y_1, \ldots, Y_5$ which are the values of the frequency response at the five selected frequency values stated in Section 7.6.1. Each frequency value was allowed its own DACE model. Initial trials in which frequency was treated as an additional factor incorporated into the experiment were not successful (see Section 6.2). (A heuristic method for selecting frequency values for simulation using cross validation is described in Chapter 6). Results are presented for each of three designs and for the four

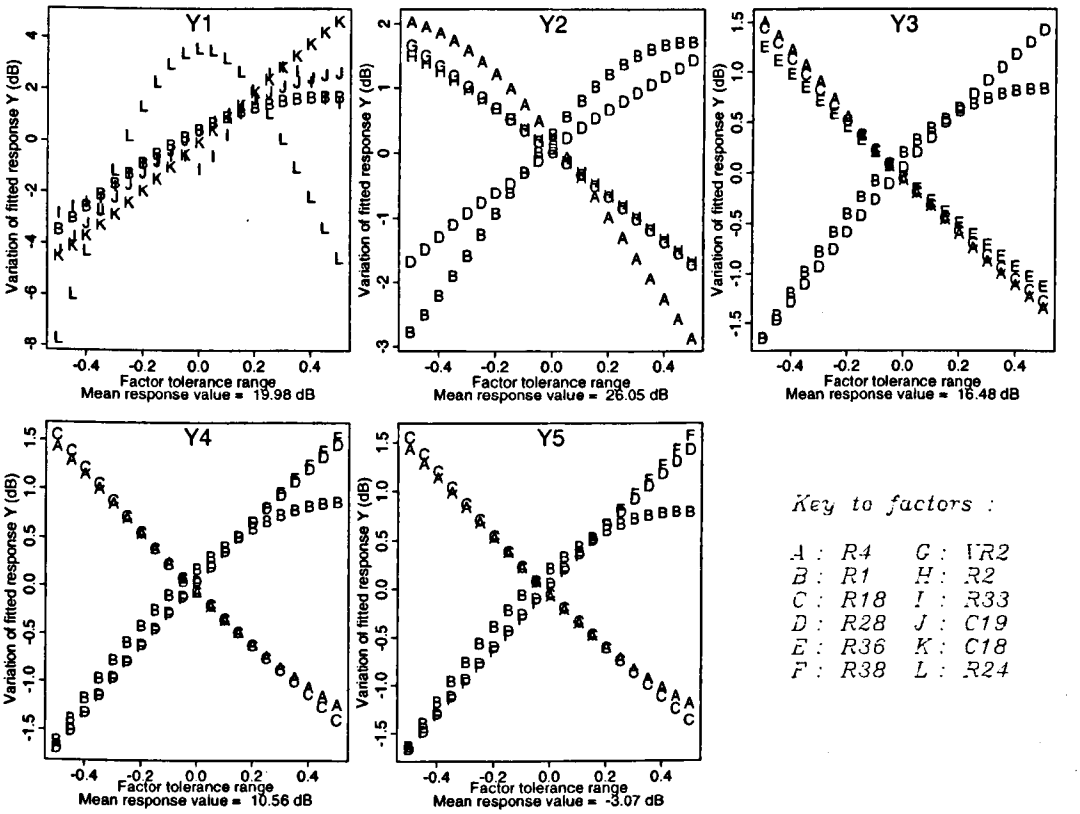


Figure 7.6: Factor plots of the 5 most important variables for the 5 responses $Y_1, \ldots, Y_5$.

combinations of restricted/unrestricted experiment and restricted/unrestricted prediction region. Factor plots (see Section 3.3.4 for an explanation of factor plots) of the 5 most important factors for each of the five models are shown in Figure 7.6 for the restricted Lattice design experiment.

Figure 7.7 shows a typical frequency response curve with the five selected values highlighted. The graphs are of *fitted* versus *actual* response for each frequency value and for the restricted Lattice design experiment. There is one frequency value, response Y_1 in Table 7.3, corresponding to the peak amplitude where the predictions are worse.

Table 7.4 shows the results of prediction selecting the 20 'most significant' factors based

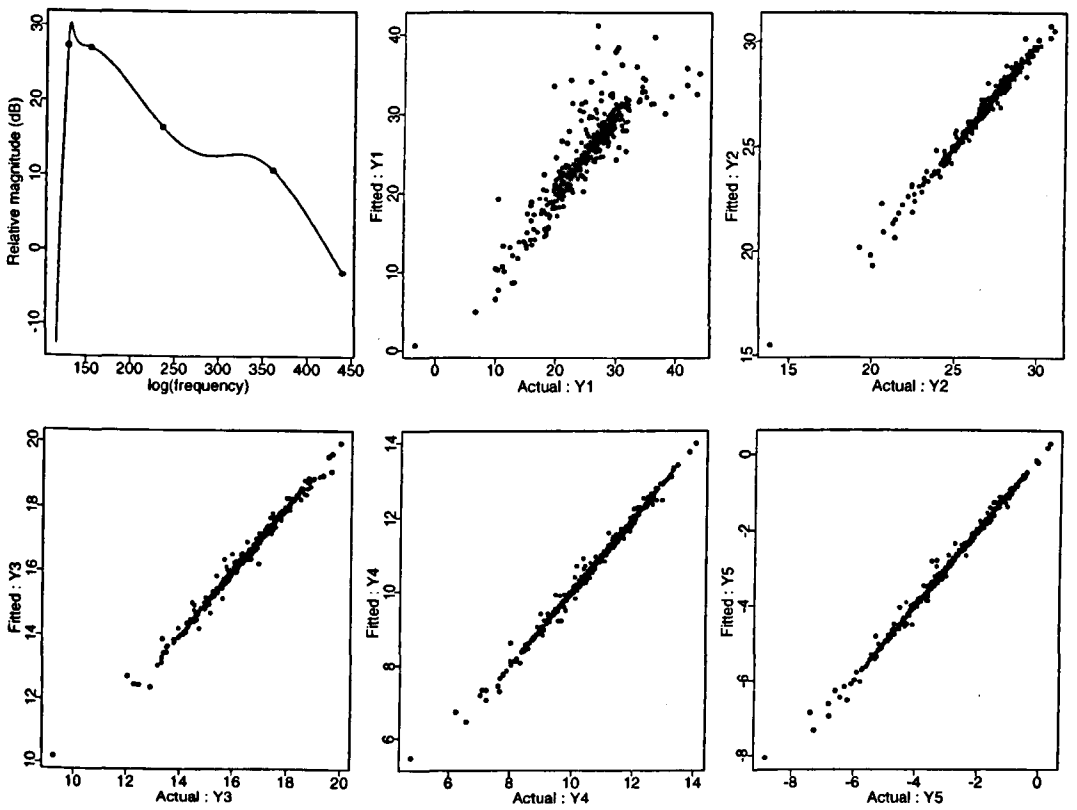


Figure 7.7: Sample points and measured vs. predicted points for the 5 responses.

Design type	Response	Full model predictions		Block model predictions	
		Full space	Restricted space	Full space	Restricted space
LHS Mk1	3	0.262	0.169	0.256	0.099
LHS Mk2	3	0.308	0.202	0.273	0.098
Lattice	3	0.383	0.198	0.483	0.224

Table 7.4: Mean ERMSE for prediction at 500 points - 20 variable model

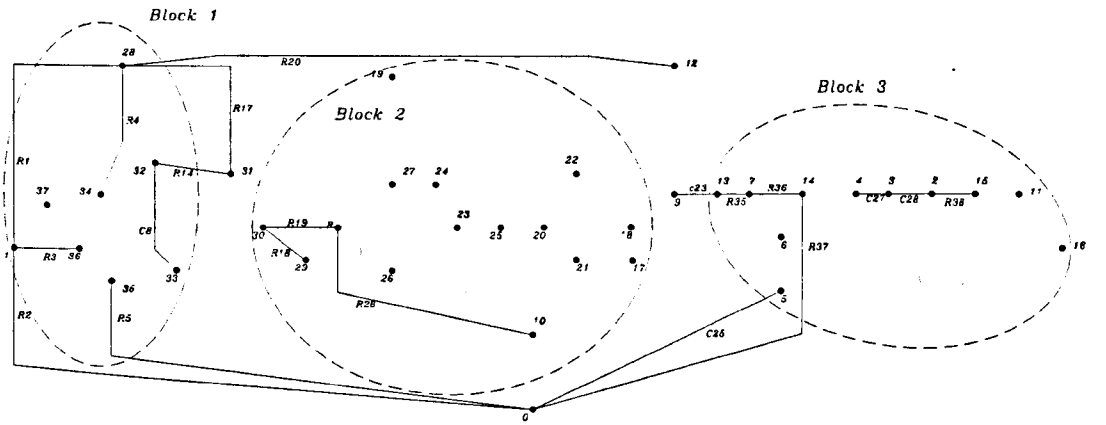


Figure 7.8: Graph showing 20 most important factors.

simply on the size of the estimates of θ_j, and only for the central frequency value. Figure 7.8 shows the subgraph containing only the significant factors (edges). It is of some interest that the significant factors for the selected response tend to form 'cliques' and that the border plays a strong connecting role. We should like to encourage the development of diagrams such as Figure 7.8 which weave together system structure and the statistical significance of components.

7.7 Conclusions

The following conclusions can be drawn.

i. Despite the high dimensionality of the problem and except perhaps for the peak frequency the emulator model is effective. For example it may be used for fast optimization and sensitivity analysis.

ii. There is an advantage in blocking, particularly if predictions are only needed over the restricted space.

iii. The single generator lattice designs work effectively for smaller dimensions and are strongly recommended as an alternative to Latin Hypercube designs. For larger dimensions it may be necessary to use more than one generator because some low dimensional projections of the design are not satisfactory when the dimension is too high relative to the sample size.

The need to experiment on large systems should lead to newer styles of experimental design and analysis in which the structure of the experiment broadly reflects the system structure. It is essential to emulate the environment in which each subsystems lives. Lattice and other easy-to-generate codes are effective on examples but there is considerable theoretical and computational work needed to establish optimality for response surface models.

By adapting a recently published algorithm to produce a BBD matrix from a sparse matrix the circuit factors, corresponding to elements in the matrix Y, can be grouped together with groups being connected with each other via factors in the matrix border. The groups of factors share the factors in the border and communicate with each other through them, Hence the analogy with Diakoptics where separate sets of equations are joined by a connecting network. Further development in the integration of sparse matrix techniques with RED should improve the efficiency of modelling systems for optimization, particularly in the area of emulator construction.

References

[1] R J Buck and H P Wynn. Improving the distribution of points in a Latin Hypercube sample. *Technometrics*, submitted.

[2] Leon O. Chua and Pen-Min Lin. *Computer Aided Analysis of Electronic Circuits : algorithms and computational techniques.* Prentice Hall Inc., Englewood Cliffs, New Jersey, 1975.

[3] Gary D Hachtel and Alberto L Sangiovanni-Vincentelli. A survey of third-generation simulation techniques. *Proceedings of the IEEE*, 69:1264–1280, Oct 1981.

[4] G Kron. *Diakoptics: The Piecewise Solution of Large Scale Systems.* MacDonald, London, 1963.

[5] M. D. McKay, W J Conover, and R J Beckman. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21:239–245, 1979.

[6] M. D. Mesarovic, D. Macko, and Y. Takahara. *Theory of Hierarchical, Multilevel Systems.* Academic Press, New York, 1970.

[7] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* CBMS-NFS, SIAM, Philadelphia, 1992.

[8] Fang K T and Wang Y. *Number-theoretic Methods in Statistics.* Chapman & Hall, London, 1994.

[9] H. P. Wynn and A. A. Zhigljavsky. *Fundamentals of Search.* Springer-Verlag, New York, to appear.

[10] A I Zečevič and D D Šiljak. Balanced decompositions of sparse systems for multilevel parallel processing. *IEEE Trans. Circuits & Systems-I: Fundamental Theory & Applications.*, 41(3):220–233, March 1994.

Chapter 8

Conclusions

There is a clear need for Robust Engineering Design methods to improve the design, manufacture and use of products and the analysis and optimization of systems in general. The application of state-of-the-art RED methods to real design problems is often hampered by the complexity of specific problems and the computer and time resources available. In this sense complexity can be defined as our ability to deal with the problem. The framework which has evolved for RED is summarised loosely as:

i. define system inputs and responses of interest

ii. design the experiment

iii. do the experiment (real or simulated)

iv. emulate the system with a simplified model

v. optimize the system emulator

vi. confirm results - repeat with reduced input space if more accuracy required

The work contained in this thesis is aimed at reducing the complexity of performing RED experiments for design optimization. This has been approached through the development of

i. a common framework for RED

ii. global optimization of designs with respect to quality.

iii. methods for modelling response functions

iv. methods for the physical decomposition of systems for RED

v. methods for reducing the input space of RED experiments

The application of these methods to electronic circuit design problems has involved the use of partitioning algorithms, system decomposition methods, simulation theory, circuit optimization, experimental design and model building to electronic circuit design problems.

The application of RED to circuit design problems is to some extent dependent on the problem itself. Different types of analysis require different approaches, in particular when systems are physically decomposed and this has been considered. The provision of software to perform RED on circuits within a commercial circuit simulation environment, using SPICE, has allowed experiments to be conducted quickly and efficiently and the collaborative project with Mentor Graphics (UK) Ltd. detailed in Chapter 3 provides a platform for this with some basic RED tools.

A novel approach to system optimization is presented in Chapter 4. The notion of quality as discussed in Chapter 1 is encapsulated in a method of global system optimization where system emulators are combined with a global numerical optimizer. The issue of

system parameter tolerances is raised and incorporated into the optimization problem.

Intelligent techniques are used for the analysis of complex systems represented by electronic circuits. Often in circuit analysis the response to be modelled is a function of time or frequency and this leads to the idea of including these as parameters in the emulator models of the system. This however proved to be impractical due to limitations in computer power and the overwhelming effect the importance of these parameters has on the emulator model obscuring the effects of the system parameters. A different approach to the integration of RED and simulation is needed. The rationale which emerged is to form a tiered arrangement of emulators to model a large system drawing from systems theory where a controller is used to direct sub-systems.

In tackling the issue of complexity in RED two decomposition strategies have been employed to simplify the problem and are categorised as *partitioning* and *tearing*, the distinction being that partitioning involves a *physical* decomposition of the system whereas tearing decomposes system parameters into a set of connected groups without affecting the physical structure of the system. Another useful distinction between the two methods is that in partitioning the system equations are formulated *after* decomposition and in tearing they are formulated *before* decomposition. Both methods use the topology of the system represented by an undirected graph as the metric for decomposition. Partitioning follows the lines of direct decomposition where sub-systems are formed and modelled independently, the emulation models being combined for optimization of the whole system at the final stage. This is particularly useful where a system is too large to be analysed whole or where different types of analysis are required for the subsystems.

Tearing provides a way of incorporating system information into the experimental design plan for more efficient experimentation and modelling for RED. The exploitation of the special design plans for more efficient emulator building is a logical next step for this work.

The examples given show that adopting the techniques presented provides a significant reduction in the complexity of performing RED on large problems.

8.1 Future work

The case studies detailed in this thesis relate to the analysis of electronic circuits. There are several issues raised in the analysis of circuits which constitute areas of future work. These include:

i. Linking non-linear device model parameters with manufacturing parameters.

ii. Dealing with feedback loops and other non-linearities in the decomposition of circuits.

iii, Mixed-mode simulation and circuit decomposition.

iv, Use of different techniques for system decomposition such as clustering algorithms and the integration of decomposition with analysis.

It should be noted however that the issues raised in (ii) and (iii) are only applicable if the circuit is physically partitioned as in Chapter 5.

On a more general level the direction of research in this thesis highlights several issues in the design of complex systems:

i. The efficient use of RED requires the integration of experimental design and model building with knowledge of the system under observation. System parameters may be difficult to change and this may have a bearing on the order of experimentation. Including engineering knowledge such as system topology or known relationships between subsystems/parameters is desirable and a framework for doing this easily would make the RED process more efficient.

ii. Expansion of the scope of RED experiments to other engineering fields, notably mechanical engineering, is required especially for the effective use of RED in product development which typically incorporates several types of engineering. The definition of system parameters in areas such as mechanical engineering is critical as this restricts the solution space of the problem. Of particular importance is the choice of geometric parameters.

iii. Intelligent techniques for dealing with complex systems need to be incorporated into real software tools which close the loop of design synthesis and analysis. The use of such tools speeds up the analysis of new design solutions and acts as a catalyst for creative design and innovation.

It is hoped that the techniques and ideas presented in this thesis prove useful in the continuing development of tools for design analysis and necessary integration of analysis and synthesis in design.

Appendix A

C routines

A.1 Improved min-cut algorithm

```
/*****************************************************************
 Algorithm copied from the Fiduccia & Mettheyeses paper.
 File mc.list contains the network information.
 File blockA.in gives the initial partition of the network.
*****************************************************************/
#include <string.h>
#include <malloc.h>
#include <stdio.h>
#define EOL '\n'
#define NODEMAX 2000
#define CELLNAMEMAX 5
#define TRUE 1
#define FALSE 0

typedef int boolean;

typedef struct { char name[CELLNAMEMAX]; } NAME;

/* define structure for BUCKET which will be 2 doubly-linked lists.*/

typedef struct dlist {
                        int dcell;
                        struct dlist *leftp;
                        struct dlist *rightp;
                        }
                        DLIST;

/* by using a lookup table to keep the cell names
   we can use the same data structures for both lists */

typedef struct list {
```

```
      int graph_ref;
      DLIST *cell_ptr;
      int block_location;
      struct list *next;
      }
      LIST;

/* set up pointers for each element in cell_array & net_array */
LIST *ca_start[NODEMAX], *ca_point[NODEMAX];
LIST *na_start[NODEMAX], *na_point[NODEMAX];
NAME lookup[NODEMAX];
DLIST *bucket_a,*bucket_b;
int *FREE_CELL_LIST,pmax,ncells,nnets;
int A_MAXGAIN,B_MAXGAIN,bal_tol;

/*-----------------------------------------------*/
/* function rb_abs computes absolute value of x.  use this      */
/* because couldn't get Sun library abs function to work        */
/*-----------------------------------------------*/
int rb_abs(int x)
{
   if(x<0) x=(-1)*x;
   return(x);
}

/*-----------------------------------------------*/
int *AllocInt(int n)
{
    int *B;
    B = ( int *) calloc(n,sizeof(int));
    return B;
}
/*-----------------------------------------------*/


/*****************************************************************/
LIST* insert(int thing, LIST *old_pointer)
{
   LIST *pointer;

   pointer = (LIST *)malloc(sizeof(LIST));
   if (pointer == NULL)
   {
      printf("Not enough memory");
      exit(1);
   }
```

```
   pointer->graph_ref = thing;
   pointer->next = old_pointer;
   return(pointer);
}

/*****************************************************************/
void printout(int total, int total2)
{
   int i=0;

   for (i=0;i<total;i++)
   {
      printf("%s ",&lookup[i]);
      for (ca_point[i]=ca_start[i];
         ca_point[i] != NULL;
         ca_point[i]=ca_point[i]->next)
      {
       printf("%5d", ca_point[i]->graph_ref);
      }
   printf("\n");
   }

        for (i=0;i<total2;i++)
        {
                printf("%3d ",i);
                for (na_point[i]=na_start[i];
         na_point[i] != NULL;
         na_point[i]=na_point[i]->next)
                {
                 printf("%s ", lookup[na_point[i]->graph_ref]);
                }
        printf("\n");
        }
}

/*****************************************************************/
/*input file syntax : # of cells & # of nets, followed by lines of
  cell names with nets they are connected to.
  NOTE : nets must be named zero to # of nets */
int input_data()
{
   NAME cell_name;
   char c;
   int i,net_value,pincount;
   int pmax=0;
```

```
   FILE* in;
   in = fopen("mc.list","r");

   fscanf(in,"%d",&ncells);
   for (i=0;i<ncells;i++) ca_start[i] = NULL;

   fscanf(in,"%d",&nnets);
   for (i=0;i<nnets;i++) na_start[i] = NULL;

   for (i=0;(c=fgetc(in))!=EOF;i++)
   {
      ungetc(c,in);
      fscanf(in,"%s",&cell_name);

      /* set up lookup table for cell names*/
      lookup[i] = cell_name;
      pincount=0;

      while ((c=fgetc(in))!=EOL)
      {
         pincount++;
         ungetc(c,in);
         fscanf(in,"%d",&net_value);

               ca_point[i] = ca_start[i];
         ca_start[i] = insert(net_value,ca_point[i]);

         na_point[net_value] = na_start[net_value];
         na_start[net_value] = insert(i,na_point[net_value]);
      }
      if (pincount>pmax) pmax=pincount;
   }
   fclose(in);
   /*printout(ncells,nnets);*/
   return(pmax);
}

/*****************************************************************/
/*use location info in cell array to count # of cells in a
specified block on a specified net  */
/*****************************************************************/
int count_cells(int block, int net)
{
   int count=0;
   LIST *net_ptr, *cell_ptr;
```

```
   net_ptr = na_start[net];
   while (net_ptr != NULL)
   {
      cell_ptr = ca_start[net_ptr->graph_ref];
      if (cell_ptr->block_location == block) count++;
      net_ptr = net_ptr->next;
   }
   return(count);
}

/*****************************************************************/
boolean balance(int movecell)
{
   int i,asum=0;
   int rtn_vec;

   if (movecell!=-1)
   {
      /***************************
       find the # of cells in block A.
      ***************************/
      for (i=0;i<ncells;i++)
         if (ca_start[i]->block_location==0)
               asum++;

      /************************************
       adjust for movement of test cell 'movecell'.
      ************************************/
      if (ca_start[movecell]->block_location==0)
               asum--;
      else asum++;

      /***********************************
      find % difference in integer form from zero
      ***********************************/
      rtn_vec=(int )(((asum*100)/ncells)-50);
      rtn_vec=rb_abs(rtn_vec);
   }
   else rtn_vec=50;
   return(rtn_vec);
}

/*****************************************************************/
int *select_cell(int block,int maxgain)
{
   int F,T,cell,net,fblk,tblk;
```

```
    int best_cell=(-1),best_f=(-1),best_t=(-1);
    DLIST *dpointer;
    LIST *pointer;
    int rtn_vec[3];

    if (maxgain!=(-pmax-1))
    {
       if (block==0) dpointer=&bucket_a[maxgain+pmax];
       else dpointer=&bucket_b[maxgain+pmax];

       while (dpointer->rightp!=NULL)
       {
          F=0;
          T=0;
          dpointer=dpointer->rightp;
          cell=dpointer->dcell;
          fblk=ca_start[cell]->block_location;
          if (fblk==0) tblk=1; else tblk=0;
          pointer=ca_start[cell];
          while (pointer!=NULL)
          {
             net=pointer->graph_ref;
             F=F+count_cells(fblk,net);
             T=T+count_cells(tblk,net);
             pointer=pointer->next;
          }
          if ((best_f==(-1)) || (F<best_f)
             || ((F==best_f)&&(T>best_t)))
          {
             best_f=F;
             best_t=T;
             best_cell=cell;
          }

       }
    }
    rtn_vec[0]=best_cell;rtn_vec[1]=best_f;rtn_vec[2]=best_t;
    return(rtn_vec);
}
/*****************************************************************/
/*Select cell to move from one block to another. Use select_cell
to get best cell from each group then use balance to pick to base cell.
In the event of a tie the cell with the best balance coeff is chosen.
*****************************************************************/

int select_base_cell_old()
```

```
{
   int best_a,best_b,gain_a,gain_b,base_cell;

   best_a=select_cell(0,A_MAXGAIN)[0];
   if (best_a!=-1) gain_a=A_MAXGAIN;
   else gain_a=(-pmax-1);

   best_b=select_cell(1,B_MAXGAIN)[0];
   if (best_b!=-1) gain_b=B_MAXGAIN;
   else gain_b=(-pmax-1);

   if ((best_a!=-1)&&(best_b!=-1))
   {
      if (gain_a>gain_b)
      {
         if (balance(best_a)<=bal_tol) base_cell=best_a;
         else base_cell=best_b;
      }
      if (gain_a<gain_b)
      {
         if (balance(best_b)<=bal_tol) base_cell=best_b;
         else base_cell=best_a;
      }
      if (gain_a==gain_b)
      {
         if (balance(best_a)<=balance(best_b)) base_cell=best_a;
         else base_cell=best_b;
      }
   }
   else if ((best_a!=-1)&&(best_b==-1)&&(balance(best_a)<=bal_tol))
               base_cell=best_a;
   else if ((best_a==-1)&&(best_b!=-1)&&(balance(best_b)<=bal_tol))
               base_cell=best_b;
   else base_cell=(-1);

   return(base_cell);
}

/*****************************************************************/
/*Select cell to move from one block to another. Use select_cell
to get best cell from each group then use balance to pick to base cell.
In case of a tie use extra info from select_cell to get '2nd order gain'
 and choose cell which is more likely to give an improvement next move.
*****************************************************************/
int select_base_cell()
{
```

```
int best_a,best_b,gain_a,gain_b,base_cell;
int Fa, Ta, Fb, Tb;

best_a=select_cell(0,A_MAXGAIN)[0];
Fa=select_cell(0,A_MAXGAIN)[1];
Ta=select_cell(0,A_MAXGAIN)[2];
if (best_a!=-1) gain_a=A_MAXGAIN;
else gain_a=(-pmax-1);

best_b=select_cell(1,B_MAXGAIN)[0];
Fb=select_cell(1,B_MAXGAIN)[1];
Tb=select_cell(1,B_MAXGAIN)[2];
if (best_b!=-1) gain_b=B_MAXGAIN;
else gain_b=(-pmax-1);

if ((best_a!=-1)&&(best_b!=-1))
{
   if (gain_a>gain_b)
   {
      if (balance(best_a)<=bal_tol) base_cell=best_a;
      else if (balance(best_b)<=bal_tol) base_cell=best_b;
   }
   if (gain_a<gain_b)
   {
      if (balance(best_b)<=bal_tol) base_cell=best_b;
      else if (balance(best_a)<=bal_tol) base_cell=best_a;
   }
   if (gain_a==gain_b)
   {
      if ((balance(best_a)<=bal_tol)&&(balance(best_b)<=bal_tol))
      {
         if ((Fa<Fb)||((Fa==Fb)&&(Ta>=Tb)))
               base_cell=best_a;
         else base_cell=best_b;
      }
      else if (balance(best_a)<=bal_tol) base_cell=best_a;
      else if (balance(best_b)<=bal_tol) base_cell=best_b;
      else base_cell=(-1);
   }
}
else if ((best_a!=-1)&&(best_b==-1)&&(balance(best_a)<=bal_tol))
        base_cell=best_a;
else if ((best_a==-1)&&(best_b!=-1)&&(balance(best_b)<=bal_tol))
        base_cell=best_b;
else base_cell=(-1);
return(base_cell);
```

```
}

/*****************************************************************/
void dremove(int cell)
{
   DLIST* dptr;

   dptr = ca_start[cell]->cell_ptr;
   dptr->leftp->rightp = dptr->rightp;
   if (dptr->rightp != NULL) dptr->rightp->leftp = dptr->leftp;
   /*free((char *)dptr);*/   /*pointer addr. is char in C++ */
   ca_start[cell]->cell_ptr = NULL;
}

/*****************************************************************/
/*place cell in dlist at new gain position*/
void move_dcell(int cell, int gain_change)
{
   DLIST *old_ptr,*dptr;

   dptr = old_ptr = ca_start[cell]->cell_ptr;

           /*point to head of new gain dlist*/
   while (dptr->leftp != NULL) dptr = dptr->leftp;
   if (gain_change==1) dptr++;
   else if (gain_change==-1) dptr--;

   if (dptr->dcell != -1)
   {
      printf("\nERROR : gain out of range.");
      printf(" Tried to change gain of %s by %d\n",
lookup[cell],gain_change);
   }
   else
   {
      /*printf("%s(%d), ",lookup[cell],gain_change);*/
      /*remove cell from old position*/
      old_ptr->leftp->rightp = old_ptr->rightp;
              if (old_ptr->rightp != NULL)
old_ptr->rightp->leftp = old_ptr->leftp;

      /*put cell at head of new position*/
      old_ptr->rightp=dptr->rightp;
      if (dptr->rightp != NULL) dptr->rightp->leftp=old_ptr;
      old_ptr->leftp=dptr;
      dptr->rightp=old_ptr;
```

```
      /*free((char *)old_ptr);*/ /*screws things up*/
   }
}

/*****************************************************************/
int max_gain_calc(int block)
{
   int count;
   DLIST *dpointer;

   count=2*pmax;
           if (block==0) dpointer=&bucket_a[count];
           else dpointer=&bucket_b[count];
   while ((dpointer->rightp==NULL)&&(count>-1))
   {
   count--;
   if (block==0) dpointer=&bucket_a[count];
   else dpointer=&bucket_b[count];
   }
   count=count-pmax;
   return(count);
}

/*****************************************************************/
void printbucket()
{
   int i;
   DLIST *dpointer;

   for (i=0;i<(2*pmax+1);i++)
   {
      dpointer=&bucket_a[i];
      while (dpointer->rightp!=NULL)
      {
         dpointer=dpointer->rightp;
         printf("A[%d],dcell=%s ",(i-pmax),lookup[dpointer->dcell]);
         if (dpointer->rightp==NULL) printf("\n");
      }
      dpointer=&bucket_b[i];
      while (dpointer->rightp!=NULL)
      {
         dpointer=dpointer->rightp;
         printf("B[%d],dcell=%s ",(i-pmax),lookup[dpointer->dcell]);
         if (dpointer->rightp==NULL) printf("\n");
      }
```

```
    }
}

/*************************************************************/
/* count the # of cut nets for the partition. 'sum' is used to
break ties between moves to find the best move for the pass.*/
int *count_cut_nets()
{
    LIST *pointer;
    int i,cut=0,cell_id,a_count,b_count;
    int sum=0,rtn_vec[2];

    for (i=0;i<nnets;i++)
    {
        a_count=b_count=0;
        pointer=na_start[i];
        do
        {
            cell_id=pointer->graph_ref;
            if (ca_start[cell_id]->block_location==0) a_count++;
            else b_count++;
            pointer=pointer->next;
        } while (pointer!=NULL);
        if ((a_count!=0)&&(b_count!=0))
        {
            cut++;
            /*printf("Net %d : A=%d, B=%d.\n",i,a_count,b_count);*/
            sum=sum+(rb_abs((a_count-b_count)));
        }
    }
    /*printf("TOTAL=%d.\n",sum);*/
    rtn_vec[0]=cut;rtn_vec[1]=sum;
    return(rtn_vec);
}

/*************************************************************/
/*****************************************
 initialize partition for blocks A and B.
*****************************************/
void partition()
{
    int i;
    int c;
    NAME tmp_cell_name;
    FILE* in;
    in = fopen("blockA.in","r");
```

```
    /* read file and update cell array locations */
    /*printf("Entering cells in Block A.\n ");*/
    while ((c=fgetc(in))!=EOL)
    {
       ungetc(c,in);
       fscanf(in,"%s",&tmp_cell_name);
       for (i=0;i<ncells;i++)
          if (strcmp((char *)&lookup[i],(char *)&tmp_cell_name)==0)
                              ca_start[i]->block_location = 0;
    }
    fclose(in);
 }
/*****************************************************************/
void change_gain1(LIST *ptr, int change)
{
   int cell;
   while (ptr != NULL)
   {
      cell=ptr->graph_ref;
      if (FREE_CELL_LIST[cell]==0)
         move_dcell(cell,change);
      ptr=ptr->next;
   }
}
/*****************************************************************/
void change_gain2(LIST *ptr, int change, int block)
{
   int cell;
                     while (ptr != NULL)
                     {
      cell=ptr->graph_ref;
      if ((FREE_CELL_LIST[cell]==0)&&
         (ca_start[ptr->graph_ref]->block_location==block))
         move_dcell(cell,change);
      ptr=ptr->next;
   }
}

/*****************************************************************/
/*****************************************************************/
int* mincut()
{
   int *rvec;
   int i,gain,selected_net;
   int gain_index;
```

```
int from_count,to_count,base_cell,nmoves=0;
int bestpass=0,ncutnets,best_cut;
int F,T,cutbal,bestcutbal;
int block;
LIST *ref_ptr;
DLIST *dref_ptr,*new_dptr;
FILE* out;

rvec=AllocInt(2);

best_cut=nnets;  /*set best_cut to a high number*/
bestcutbal=nnets;
block=0;
A_MAXGAIN = -pmax;
B_MAXGAIN = -pmax;

for (i=0;i<ncells;i++) FREE_CELL_LIST[i]=0;    /*0=free*/

        for (i=0;i<(2*pmax+1);i++)
{
   bucket_a[i].rightp = bucket_a[i].leftp = NULL;
   bucket_b[i].rightp = bucket_b[i].leftp = NULL;
   bucket_a[i].dcell = bucket_b[i].dcell = -1;
}

/*****************************************
  need to define an (A,B) cell distribution,
  i.e an initial split.
*****************************************/
/* initialize all cells to block B */
for (i=0;i<ncells;i++) ca_start[i]->block_location = 1;

partition();

/*****************************************
  compute initial gains for each cell given
  block_location info.
*****************************************/
for (i=0;i<ncells;i++)
{
   gain = 0;
   ref_ptr = ca_start[i];
   F = ref_ptr->block_location;
   if (F==0) T=1; else T=0;
   while (ref_ptr != NULL)
   {
```

```
        selected_net = ref_ptr->graph_ref;
        if (count_cells(F,selected_net)==1) gain++;
        if (count_cells(T,selected_net)==0) gain--;
        ref_ptr = ref_ptr->next;
      }

      /*compute MAXGAIN for both block A & B*/
      switch(F)
      {
         case 0 : if (gain>A_MAXGAIN) A_MAXGAIN=gain;
            break;
                              case 1 : if (gain>B_MAXGAIN)
B_MAXGAIN=gain;
                                          break;
      }

      /*printf(" gain=%d ,Lookup[i]=%s \n",gain,lookup[i]);*/
      /************************************
        add cell i to bucket A(0) or B(1)
          (corresponding to its location)
        at position [gain]
      ************************************/
      gain_index = gain + pmax;  /* alters index from +-gain*/
      if (F==0) dref_ptr = &bucket_a[gain_index];
      else dref_ptr = &bucket_b[gain_index];

      new_dptr = (DLIST *)malloc(sizeof(DLIST));
              if (new_dptr == NULL)
              {
                        printf("Not enough memory");
                        exit(1);
              }
      new_dptr->dcell = i;
                     if (dref_ptr->rightp != NULL)
                          dref_ptr->rightp->leftp = new_dptr;
      new_dptr->rightp = dref_ptr->rightp;
      dref_ptr->rightp = new_dptr;
      new_dptr->leftp = dref_ptr;

      /*****************************************
       for each cell in bucket need to point to it
         from cell array!!
      *****************************************/
      ca_start[i]->cell_ptr = new_dptr;
   }
   /*printf("%d %d\n",A_MAXGAIN,B_MAXGAIN);*/
```

```
/**********************************
 select cell to move from one block
 to another. Use balance() to
 find best cell in group then
 select block A or B. Then move cell
 with highest gain.
**********************************/
base_cell=select_base_cell();
while (base_cell!=-1)
{
   nmoves++;
   /*printf("****************move %d***************\n",nmoves);
   printbucket();
   printf("New MAXGAIN for A=%d, New MAXGAIN for B=%d\n",
         A_MAXGAIN,B_MAXGAIN);

   printf("base cell=%d\n",base_cell);
   printf("lookup[base cell]=%s\n",lookup[base_cell]);*/

   /*********************************************
    define 'from' and 'to' blocks here but do not
    change base cell location until after gain
    adjustment.
   *********************************************/
   F=ca_start[base_cell]->block_location;
           if (F==0) T=1; else T=0;

   /**********************************
    remove base_cell from bucket list
    and place it on free cell list.
   **********************************/
   dremove(base_cell);
   FREE_CELL_LIST[base_cell]=1;

   /**********************************
    recompute cell gains with the move
    of base_cell taken into account.
    - first work on nets which are
    critical before the move....
   **********************************/
   ref_ptr = ca_start[base_cell];
   while (ref_ptr != NULL)
   {
      selected_net = ref_ptr->graph_ref;
      /*printf("Cell Gain Changed On Net %d For : ",selected_net);*/
```

```
            from_count=count_cells(F,selected_net);
            to_count=count_cells(T,selected_net);
            /*printf("F(%d) = %d.  T(%d) = %d.\n",
               selected_net,from_count,selected_net,to_count);*/

            if (to_count==0) change_gain1(na_start[selected_net],1);
            else if (to_count==1) change_gain2(na_start[selected_net],-1,T);

            /************************************
                     ... Now simulate move and work on nets
                     critical now.
                    ************************************/
            from_count--;
            to_count++;

            if (from_count==0) change_gain1(na_start[selected_net],-1);
            else if (from_count==1)
change_gain2(na_start[selected_net],1,F);

            ref_ptr = ref_ptr->next;
         }
         /*******************************************
         Now change block location marker of
         base cell in cell array.
         *******************************************/
                ca_start[base_cell]->block_location=T;

                        /*******************************************
                         recompute MAXGAIN for each block.
                        *******************************************/
                        A_MAXGAIN=max_gain_calc(0);
                        B_MAXGAIN=max_gain_calc(1);

         /*******************************************
          find the # of nets that are cut with the new
          partition.
         *******************************************/
         ncutnets=count_cut_nets()[0];
         cutbal=count_cut_nets()[1];
/*used to find best split in pass if gains tie*/
         if ((ncutnets<best_cut)||
((ncutnets==best_cut)&&(cutbal<=bestcutbal)))
         {
            best_cut=ncutnets;
            bestpass=nmoves;
            bestcutbal=cutbal;
```

```
      out = fopen("blockA.in","w");
      for (i=0;i<ncells;i++)
         if (ca_start[i]->block_location == 0)
            fprintf(out," %s",lookup[i]);
      fprintf(out,"\n");
      fclose(out);
   }

   /*********************************************
    print out results of this move.
   *********************************************/
   /*printout in group order*/
   /*printf("\n");
   printf("# of moves : %d. Moved cell = %s.  # of cut nets = %d\n",
         nmoves,lookup[base_cell],ncutnets);
   printf("-------------------------------\n");
   printf("GROUP A : ");
   for (i=0;i<ncells;i++)
      if (ca_start[i]->block_location == 0)
         printf("%s ",lookup[i]);
                  printf("\n");
                  printf("GROUP B : ");
                  for (i=0;i<ncells;i++)
                        if (ca_start[i]->block_location == 1)
                              printf("%s ",lookup[i]);
                  printf("\n");*/

   /*********************************************
    find best cells to move next from each block
   *********************************************/
   base_cell=select_base_cell();
  }
  printf("The best move was #%d, with %d net(s) cut.\n",
bestpass,best_cut);
  rvec[0]=bestpass;
  rvec[1]=best_cut;
  return(rvec);
}
/*****************************************************************************/
/*****************************************************************************/
main()
{
  int *best,*nextbest;
  int npass=1;
  FILE *out;
  best=AllocInt(2);
```

```
   nextbest=AllocInt(2);

   printf("Balance ratio (0-50):");
/*between 0 and 50. (+-% tolerance allowed)*/
   scanf("%d",&bal_tol);
   printf("\n");
   pmax=input_data();

          bucket_a = (DLIST *)malloc(((2*pmax)+1)*sizeof(DLIST));
         bucket_b = (DLIST *)malloc(((2*pmax)+1)*sizeof(DLIST));
   FREE_CELL_LIST = (int *)malloc(ncells*sizeof(int));

   printf("#PASS %d  #\n",npass);
   nextbest[0]=0;nextbest[1]=0;
   best=mincut();
   while ((best[0]!=ncells)&&
((best[0]!=nextbest[0])||(best[1]!=nextbest[1])))
   {
      npass++;
      printf("#PASS %d  #\n",npass);
      nextbest=best;
      best=mincut();
   }
   out=fopen("new2.out","a");
   fprintf(out,"%d %d %d\n",npass,best[0],best[1]);
   fclose(out);
}
```

A.2 Sparse matrix decomposition algorithm

```
/*--------------------------------------*/
Construct incidence matrix from SPICE circuit description
Data structures copied from the min-cut program.
File mc.list contains the network information.
/*--------------------------------------*/
#include <stdio.h>
#include<malloc.h>
#define bal_ratio 0.35       /*between 0 and 0.5 */
#define EOL '\n'
#define NODEMAX 70
#define CELLNAMEMAX 5
#define BORDER -1

typedef int boolean;
typedef struct {
       char name[CELLNAMEMAX];
```

```
        } Name;

typedef struct list {
            int label;
            struct list *next;
            } List;

typedef struct block {
            int name;
            int size;
            struct block *nextb;
            List *nets;
            } Block;

typedef struct vertex {
                    int name;
                    int blksum;
                    int netsum;
                    Block *blocks;
                    List *nets;
                    struct vertex *nextv;
                    } Vertex;

/* set up pointers for each element in cell_array & net_array */
Block *CA_START,*NA_START,*SC_START;
Block *TC_START,*GP_START,*CUT_START;

Vertex *V_START;
Block *B_START;

/* use LOOKUP table to keep the cell names*/
Name *LOOKUP;

int NCELLS,NNETS;

/*Bob's Memory Routines---------------------*/
int *AllocInt(int n)
{
    int *B;
    B = ( int *) calloc(n,sizeof(int));
    return B;
}
/*-------------------------------------*/
int **AllocInt2(int n, int p)
{
    int i;
```

```
    int **A;
    A = (int **) calloc(n , sizeof(int *));
    for(i=0;i<n;i++) A[i]=AllocInt(p);
    return A;
}
/*-------------------------------------*/
char *AllocChar(int n)
{
    char *B;
    B = ( char *) calloc(n,sizeof(char));
    return B;
}
/*-------------------------------------*/
Name *AllocName(int n)
{
    Name *B;
    B = ( Name *) calloc(n,sizeof(Name));
    return B;
}
/*-------------------------------------*/
List* n_point_to(int name, List *start)
{
   while (start!=NULL)
   {
      if (start->label==name) return(start);
      start=start->next;
   }
   return(start);
}
/*-------------------------------------*/
Block* b_point_to(int name, Block *start)
{
   while (start!=NULL)
   {
     if (start->name==name) return(start);
      start=start->nextb;
   }
   return(start);
}
/*-------------------------------------*/
Vertex* v_point_to(int name, Vertex *start)
{
   while (start!=NULL)
   {
      if (start->name==name) return(start);
      start=start->nextv;
```

```
    }
    return(start);
}
/*-------------------------------------*/
List* insert(int thing, List *old_pointer)
{
      List *pointer;

      pointer = (List *)malloc(sizeof(List));
      if (pointer == NULL)
      {
            printf("Not enough memory");
            exit(1);
      }
      pointer->label = thing;
      pointer->next = old_pointer;
      return(pointer);
}
/*-------------------------------------*/
Vertex* vinsert(int thing, Vertex *old_vptr)
{
      Vertex *vptr;

      vptr = (Vertex *)malloc(sizeof(Vertex));
      if (vptr == NULL)
      {
            printf("Not enough memory");
            exit(1);
      }
      vptr->name = thing;
      vptr->nextv = old_vptr;
      vptr->nets = NULL;
      vptr->blocks = NULL;
      return(vptr);
}
/*-------------------------------------*/
Block* binsert(int name,int size, Block *old_bptr)
{
      Block *bptr;

      bptr = (Block *)malloc(sizeof(Block));
      if (bptr == NULL)
      {
            printf("Not enough memory");
            exit(1);
      }
```

```
        bptr->name = name;
        bptr->size = size;
        bptr->nextb = old_bptr;
        bptr->nets = NULL;
        return(bptr);
}
/*-------------------------------------*/
void nvremove(Vertex *start,int net)
{
   List *ptr,*prevptr;

   ptr=start->nets;
   while ((ptr->label!=net)&&(ptr!=NULL))
   {
      prevptr=ptr;
      ptr=ptr->next;
   }
   if (ptr==start->nets) start->nets=ptr->next;
   else prevptr->next=ptr->next;
   free((char *)ptr);
}
/*-------------------------------------*/
void remove(Block *bptr,int net)
{
   List *start,*prevptr,*ptr;

   start=bptr->nets;
   prevptr=NULL;
   ptr=start;
   while ((ptr->label!=net)&&(ptr!=NULL))
   {
      prevptr=ptr;
      ptr=ptr->next;
   }
   if (prevptr!=NULL) prevptr->next=ptr->next;
   else bptr->nets=ptr->next;
   free((char *)ptr);
}
/*-------------------------------------*/
Block* bremove(Block *bstart,int name)
{
   Block *prevbptr,*bptr;

   prevbptr=NULL;
   bptr=bstart;
   while ((bptr->name!=name)&&(bptr->nextb!=NULL))
```

```
    {
       prevbptr=bptr;
       bptr=bptr->nextb;
    }
    if (bptr->nets!=NULL) free((char *)bptr->nets);
    if (prevbptr!=NULL) prevbptr->nextb=bptr->nextb;
    else
    {
       bptr=bstart;
       bstart=bstart->nextb;
    }
    free((char *)bptr);
    return(bstart);
}
/*-------------------------------------*/
Vertex* vremove(Vertex *vstart,int net)
{
   Vertex *prevptr,*vptr;

   prevptr=NULL;
   vptr=vstart;
   while ((vptr->name!=net)&&(vptr!=NULL))
   {
      prevptr=vptr;
      vptr=vptr->nextv;
   }
   if (vptr->nets!=NULL) free((char *)vptr->nets);
   if (vptr->blocks!=NULL) free((char *)vptr->blocks);
   if (prevptr!=NULL) prevptr->nextv=vptr->nextv;
   else
   {
      vptr=vstart;
      vstart=vstart->nextv;
   }
   free((char *)vptr);
   return(vstart);
}
/*-------------------------------------*/
Block* copy(Block *start)
{
   Block *bptr,*newstart;
   List *ptr,*newptr;

   newstart=NULL;
   while(start!=NULL)
   {
```

```
        bptr=newstart;
        newstart=binsert(start->name,start->size,bptr);
        for(ptr=start->nets;ptr!=NULL;ptr=ptr->next)
        {
              newptr=newstart->nets;
              newstart->nets=insert(ptr->label,newptr);
        }
        start=start->nextb;
    }
    return(newstart);
}
/*-------------------------------------*/
void sc_init()
{
    int i=1;
    /*need to do a -1 to get range of nets 0->NNETS-1*/
    int size=0;
    Block *hptr,*nstart;

    SC_START=NULL;
    for (nstart=NA_START;nstart!=NULL;nstart=nstart->nextb)
    {
       hptr=SC_START;
       SC_START=binsert((NNETS-i),size,hptr);
       i++;
    }
}

/*-------------------------------------*/
void printout(int total, int total2)
{
      Block *start;
      List *cptr,*nptr;
      int i=0;

      start=CA_START;
      for (i=0;i<total;i++)
      {
            printf("%s ",&LOOKUP[i]);
            for (cptr=start->nets;
                  cptr != NULL;
                  cptr=cptr->next)
            {
             printf("%5d", cptr->label);
            }
      printf("\n");
```

```
        start=start->nextb;
        }

        start=NA_START;
        for (i=0;i<total2;i++)
        {
              printf("%3d ",i);
              for (nptr=start->nets;
                    nptr != NULL;
                    nptr=nptr->next)
              {
              printf("%s ", LOOKUP[nptr->label]);
              }
        printf("\n");
        start=start->nextb;
        }
}

/*-------------------------------------*/
void printlist(List *ptr)
{
   while (ptr!=NULL)
   {
      printf("%4d",(ptr->label));
      ptr=ptr->next;
   }
   printf("\n");
}
/*-------------------------------------*/
void printblock(Block *start,char *lab)
{
   Block *bptr;

   for(bptr=start;bptr!=NULL;bptr=bptr->nextb)
   {
      printf("%s %4d %4dn:",lab,(bptr->name),(bptr->size));
      printlist(bptr->nets);
   }
}
/*-------------------------------------*/
void printvertex(Vertex *start,char *lab)
{
   Vertex *vptr;

   printf("\n");
   for(vptr=start;vptr!=NULL;vptr=vptr->nextv)
```

```
   {
      printf("%s %4d %4d %4d :",
lab,(vptr->name),vptr->blksum,vptr->netsum);
      printlist(vptr->nets);
      printblock(vptr->blocks,"block ");
   }
}

/*--------------------------------------*/
/*input file syntax : # of cells, # of nets, then lines of cell
  names with nets they are connected to.
  NOTE : nets named zero to # of nets*/
void input_data()
{
   Block *cstart,*nstart,*bptr;
   List *cptr,*nptr;
   Name cell_name;
   char c;
   int i,net_value;

   FILE* in;
   in = fopen("mc.list","r");

   fscanf(in,"%d",&NCELLS);
   CA_START = NULL;
   for (i=0;i<NCELLS;i++)
   {
      bptr=CA_START;
      CA_START=binsert((NCELLS-i-1),0,bptr);
      /*(-1) for range 0-NCELLS*/
   }

   fscanf(in,"%d",&NNETS);
   NA_START = NULL;
   for (i=0;i<NNETS;i++)
   {
      bptr=NA_START;
      NA_START=binsert((NNETS-i-1),0,bptr);
      /*(-1) for range 0-NNETS*/
   }

   LOOKUP=AllocName(NCELLS);

   i=0;
   while ((c=fgetc(in))!=EOF)
   for (cstart=CA_START;cstart!=NULL;cstart=cstart->nextb)
```

```
    {
       ungetc(c,in);
       fscanf(in,"%s",&cell_name);
       /* set up LOOKUP table for cell names*/
       LOOKUP[i] = cell_name;

       while ((c=fgetc(in))!=EOL)
       {
             ungetc(c,in);
             fscanf(in,"%d",&net_value);
             /*adjust range [1 to NNETS] to [0 to (NNETS-1)] */
             //net_value--;

                   cptr = cstart->nets;
             cstart->nets = insert(net_value,cptr);

             nstart=b_point_to(net_value,NA_START);
             nptr = nstart->nets;
                           nstart->nets = insert(i,nptr);
       }
       i++;
         }
}
/*--------------------------------------*/
void fprintmat(int **mat)
{
    int x,y;
    FILE* out;

    out=fopen("lookup","w");
    for (x=0;x<NCELLS;x++) fprintf(out,"%s ",LOOKUP[x]);
    fprintf(out,"\n");
    fclose(out);

    out=fopen("viewmat","w");
    for (x=0;x<NNETS;x++)
    {
        fprintf(out,"\n");
      for (y=0;y<NNETS;y++)
      {
            if (mat[x][y]==0) fprintf(out,"%3d ",mat[x][y]);
            else fprintf(out,"%s ",LOOKUP[mat[x][y]-1]);
      }
    }
    fprintf(out,"\n");
    fclose(out);
```

```
    out=fopen("smat","w");
    for (x=0;x<NNETS;x++)
    {
        fprintf(out,"\n");
      for (y=0;y<NNETS;y++) fprintf(out,"%2d ",mat[x][y]);
    }
    fprintf(out,"\n");
    fclose(out);
}
/*--------------------------------------*/
int listlength(int group, Block *start)
{
   int size=0;
   List *ptr;
   Block *bptr;

   bptr=b_point_to(group,start);
   for (ptr=bptr->nets;ptr!=NULL;ptr=ptr->next) size++;
   return(size);
}
/*--------------------------------------*/
/*count cells on each net and form group of
  border nets (group=-1)*/
int *getord(int dmin)
{
   int *dvec,size=0;
   List *gptr;
   Block *nstart;

   GP_START=NULL;
   GP_START=binsert(-1,size,NULL);
   dvec=AllocInt(NNETS);
   for (nstart=NA_START;nstart!=NULL;nstart=nstart->nextb)
   {
      dvec[nstart->name]=listlength(nstart->name,NA_START);
      if (dvec[nstart->name]>=dmin)
      {
            gptr=GP_START->nets;
            GP_START->nets=insert(nstart->name,gptr);
            GP_START->size++;
      }
   }
   return(dvec);
}
```

```
/*-------------------------------------*/
/* use cell & net lists to get dmat and SC_LIST */
void makesc(int **dmat)
{
   int x,y;
   List *cptr,*nptr,*scptr;
   Block *nstart,*cstart,*scstart;

   for (x=0;x<NNETS;x++)
         for (y=0;y<NNETS;y++)
         {
               if (x==y) dmat[x][y]=0;
               else dmat[x][y]=9;
         }

   sc_init();      /*get list SC ready to fill in*/
   for (nstart=NA_START;nstart!=NULL;nstart=nstart->nextb)
   {
         for (nptr=nstart->nets;nptr!=NULL;nptr=nptr->next)
         {
            cstart=b_point_to(nptr->label,CA_START);
               for (cptr=cstart->nets;cptr!=NULL;cptr=cptr->next)
               {
                     x=nstart->name;
                     y=cptr->label;
                  if (x!=y)
                  {
                        dmat[x][y]=1;
                        scstart=b_point_to(x,SC_START);
                        scptr=scstart->nets;
                        if (n_point_to(y,scptr)==NULL)
                              scstart->nets=insert(y,scptr);
                  }
            }
      }
   }

}

/*-------------------------------------*/
void maketc(int dmin,int *dvec)       /*make TC from SC*/
{
   Block *tcptr;
   List *ptr;

   TC_START=SC_START;
```

```
    for (tcptr=TC_START;tcptr!=NULL;tcptr=tcptr->nextb)
    {
       if (dvec[tcptr->name]>=dmin)
            TC_START=bremove(TC_START,tcptr->name);
       else
       {
            for(ptr=tcptr->nets;ptr!=NULL;ptr=ptr->next)
            {
                 if (dvec[ptr->label]>=dmin)
                      remove(tcptr,ptr->label);
            }
       }
    }
}
/*--------------------------------------*/
Block* locate(int net,Block *start)
{
   while (start!=NULL)
   {
      if (n_point_to(net,start->nets)!=NULL) return(start);
      start=start->nextb;
   }
   return(start);
}
/*--------------------------------------*/
Block *incutlist(int a,int b)
{
   Block *cutptr;
   cutptr=CUT_START;
   while (cutptr!=NULL)
   {
      if (cutptr->name==a) if (cutptr->nets->label==b) break;
      if (cutptr->name==b) if (cutptr->nets->label==a) break;
      cutptr=cutptr->nextb;
   }
   return(cutptr);
}
/*--------------------------------------*/
void makegroups(int size)
{
   Block *tcptr,*gptr,*gptr2,*gstart,*cutptr;
   List *ptr,*ptr2;
   int net1,net2,gpsize,group,gpcount=0;

   CUT_START=NULL;
   for(tcptr=TC_START;tcptr!=NULL;tcptr=tcptr->nextb)
```

```
{
   net1=tcptr->name;
   gstart=locate(net1,GP_START);
   if (gstart==NULL)
   {
         for(ptr=tcptr->nets;ptr!=NULL;ptr=ptr->next)
         {
               net2=ptr->label;
               gptr=locate(net2,GP_START);
               if (gptr!=NULL)
               {
                  if ((gptr->size<size)&&
                     (locate(net1,GP_START)==NULL))
                  {
                     gstart=gptr;
                     ptr2=gstart->nets;
                     gstart->nets=insert(net1,ptr2);
                     gstart->size++;
                  }
                  else
                  {
                     printf("%d+--%d cut\n",(net1),(net2));
                     gptr=CUT_START;
                     CUT_START=binsert(net1,0,gptr);
                     ptr2=CUT_START->nets;
                     CUT_START->nets=insert(net2,ptr2);
                  }
               }
         }
         if (gstart==NULL)
         {
               gptr=GP_START;
               gpsize=1;
               GP_START=binsert(gpcount,gpsize,gptr);
               gpcount++;
               ptr=GP_START->nets;
               GP_START->nets=insert(net1,ptr);
               gstart=GP_START;
         }
   }

   for(ptr=tcptr->nets;ptr!=NULL;ptr=ptr->next)
   {
         net2=ptr->label;
         gptr=locate(net2,GP_START);
         if (gptr==NULL)
```

```
        {
            if (gstart->size<size)
            {
              ptr2=gstart->nets;
              gstart->nets=insert(net2,ptr2);
              gstart->size++;
            }
            else if (incutlist(net1,net2)==NULL)
            {
              printf("%d-+-%d cut\n",(net1),(net2));
              gptr=CUT_START;
              CUT_START=binsert(net1,0,gptr);
              ptr2=CUT_START->nets;
              CUT_START->nets=insert(net2,ptr2);
            }
        }
        else if ((gptr->name!=gstart->name)&&
                    (incutlist(net1,net2)==NULL))
        {
              printf("%d--+%d cut\n",(net1),(net2));
              gptr=CUT_START;
              CUT_START=binsert(net1,0,gptr);
              ptr2=CUT_START->nets;
              CUT_START->nets=insert(net2,ptr2);
        }

   }

}
printblock(GP_START,"Pre-Cut ");
/*now adjust groups for cuts*/
for(cutptr=CUT_START;cutptr!=NULL;cutptr=cutptr->nextb)
{
   net1=cutptr->name;
   gptr=locate(net1,GP_START);
   if (gptr!=NULL)
   {
         group=gptr->name;
         /*remove net1 from group*/
         remove(gptr,net1);
         gptr->size--;
         /*if group now empty, remove group and update names*/
         if (gptr->nets==NULL)
         {
            GP_START= bremove(GP_START,group);
            for(gptr2=GP_START;gptr2->name>group;gptr2=gptr2->nextb)
```

```
                                gptr2->name--;
                }
        }
        /*put net1 into BORDER group*/
        gptr=b_point_to(BORDER,GP_START);
        ptr2=gptr->nets;
        gptr->nets=insert(net1,ptr2);
        gptr->size++;
    }

}

/*--------------------------------------*/
void buildvertex()
{
    int net,net2,group;
    Vertex *vptr;
    Block *bptr;
    Block *gptr,*scptr;
    List *ptr,*ptr2,*ptr3;

    V_START=NULL;
    B_START=NULL;
    /*for each group (excl. border group) create a block structure*/
    for(gptr=GP_START;gptr!=NULL;gptr=gptr->nextb)
    if(gptr->name!=BORDER)
    {
        bptr=B_START;
        B_START=binsert(gptr->name,gptr->size,bptr);
    }
    gptr=b_point_to(BORDER,GP_START);
    /*for each net in border ....*/
    for(ptr=gptr->nets;ptr!=NULL;ptr=ptr->next)
    {
        net=ptr->label;
        vptr=V_START;
        V_START=vinsert(net,vptr);
        V_START->netsum=0;
        scptr=b_point_to(net,SC_START);
        /*for each net2 connected to net in border....*/
        for(ptr2=scptr->nets;ptr2!=NULL;ptr2=ptr2->next)
        {
              net2=ptr2->label;
              gptr=locate(net2,GP_START);
              group=gptr->name;
              if (group==BORDER) /*enter net in vertex list (if not there)*/
```

```
            {
               ptr3=V_START->nets;
               if (n_point_to(net2,ptr3)==NULL)
               {
                  V_START->nets=insert(net2,ptr3);
                  V_START->netsum++;
               }
            }
            else
            {     /*enter net in block list (if not in already)*/
                  bptr=b_point_to(group,B_START);
                  ptr3=bptr->nets;
                  if (n_point_to(net,ptr3)==NULL)
                        bptr->nets=insert(net,ptr3);
                  /*enter block in vertex list (if not in already)*/
                  bptr=V_START->blocks;
                  if (b_point_to(group,bptr)==NULL)
                  {
                       V_START->blocks=binsert(group,gptr->size,bptr);
                       V_START->blksum+=gptr->size;
                  }

            }
      }
   }
}
/*-------------------------------------*/
void borderbalance()
{
   int s=0,net,bordernet,basename,blksize,oldblock,newblksize;
   List *ptr,*ptr2,*nptr;
   Block *gptr,*gmerge;
   Block *bbase,*bmerge,*bptr,*bptr2;
   Vertex *vptr,*vptr2,*mvptr;

   /*select vertex (net) to move from border & point to it with mvptr */
   mvptr=NULL;
   for(vptr=V_START;vptr!=NULL;vptr=vptr->nextv)
   {
      if(vptr->blocks!=NULL)
      if ((mvptr==NULL)||(vptr->blksum<mvptr->blksum)||
           ((vptr->blksum==mvptr->blksum)&&
              (vptr->netsum<mvptr->netsum)))
      mvptr=vptr;
   }
   /*merge blocks if neccessary.....*/
```

```
bordernet=mvptr->name;
bbase=mvptr->blocks;
if (mvptr!=NULL)
{
   basename=bbase->name;
   if (bbase->nextb!=NULL)
   {
   /* merge other blocks 'bmerge' in vertex list with 'bbase'....*/
   bmerge=bbase->nextb;
   while (bmerge!=NULL)
   {
         /*merge B_LIST's */
         oldblock=bmerge->name;
         bptr=b_point_to(basename,B_START);
         bptr2=b_point_to(oldblock,B_START);
         bptr->size+=bptr2->size;
         newblksize=bptr->size;
         for(nptr=bptr2->nets;nptr!=NULL;nptr=nptr->next)
            if (n_point_to(nptr->label,bptr->nets)==NULL)
            {
               ptr=bptr->nets;
               bptr->nets=insert(nptr->label,ptr);
            }

         /* for every net in unified block update vertex block list */
         bptr=b_point_to(oldblock,B_START);
         for(nptr=bptr->nets;nptr!=NULL;nptr=nptr->next)
         {
               net=nptr->label;
               vptr=v_point_to(net,V_START);
               bptr2=b_point_to(basename,vptr->blocks);
               if (bptr2!=NULL)
               {
                  vptr->blocks=bremove(vptr->blocks,oldblock);
                  /*if changing vertex block list of basename
                     then change bmerge to stop 'for' loop if needed*/
                  if ((net==basename)&&
                     (vptr->blocks->nextb==NULL)) bmerge=NULL;
               }
               else
               {
                  bptr2=b_point_to(oldblock,vptr->blocks);
                  bptr2->name=basename;
               }
         }
         /*update GP_LIST structure*/
```

```
        gptr=b_point_to(basename,GP_START);
        gmerge=b_point_to(oldblock,GP_START);
        for(ptr=gmerge->nets;ptr!=NULL;ptr=ptr->next)
        {
              if (n_point_to(ptr->label,gptr->nets)==NULL)
              {
                 ptr2=gptr->nets;
                 gptr->nets=insert(ptr->label,ptr2);
              }
        }

        B_START=bremove(B_START,oldblock);
        GP_START=bremove(GP_START,gmerge->name);
        if (bmerge!=NULL) bmerge=bmerge->nextb;
}
blksize=newblksize+1;
}
else blksize=bbase->size+1;
/*(+1) for the border net moving to the block*/

/* update B_LIST */
bptr=b_point_to(basename,B_START);
for(nptr=mvptr->nets;nptr!=NULL;nptr=nptr->next)
if (n_point_to(nptr->label,bptr->nets)==NULL)
      bptr->nets=insert(nptr->label,bptr->nets);
nptr=n_point_to(bordernet,bptr->nets);
if (nptr!=NULL) remove(bptr,bordernet);
bptr->size++;

/* update border net lists of V_LIST - del. moving net from lists*/
for(vptr=V_START;vptr!=NULL;vptr=vptr->nextv)
if (n_point_to(bordernet,vptr->nets)!=NULL)
{
      nvremove(vptr,bordernet);
      /*update Qi*/
      vptr->netsum--;
}

/*update block lists of V_LIST */
bptr=b_point_to(basename,B_START);
for(ptr=bptr->nets;ptr!=NULL;ptr=ptr->next)
{
vptr2=v_point_to(ptr->label,V_START);
bptr2=b_point_to(basename,vptr2->blocks);
if (bptr2==NULL)
       vptr2->blocks=binsert(basename,blksize,vptr2->blocks);
```

```
        else bptr2->size=blksize;
        vptr2->blksum=0;
        /*update Si*/
        for(bptr2=vptr2->blocks;bptr2!=NULL;bptr2=bptr2->nextb)
                    vptr2->blksum+=bptr2->size;
        }

        /*add border net to block in G_LIST*/
        gptr=b_point_to(basename,GP_START);
        ptr=gptr->nets;
        gptr->nets=insert(bordernet,ptr);
        gptr->size=blksize;

        /*remove net from border group (BORDER) in G_LIST*/
        gptr=b_point_to(BORDER,GP_START);
        remove(gptr,bordernet);
        gptr->size--;

        V_START=vremove(V_START,bordernet);
    }
    else
    {
        printf("ERROR - can't select a border net to move,");
        printf(" printing groups so far...\n");
        printblock(GP_START,"Group");
        exit(1);
    }
}
/*--------------------------------------*/
void writematrix(Block *start)
{
    int *order,*inverseorder,i,cell,net1,net2,**smat;
    Block *bptr,*cptr;
    List *ptr,*ptr2;

    order=AllocInt(NNETS);
    inverseorder=AllocInt(NNETS);
    smat=AllocInt2(NNETS,NNETS);
    i=NNETS;
    for(bptr=start;bptr!=NULL;bptr=bptr->nextb)
    {
       ptr=bptr->nets;
          while (ptr!=NULL)
          {
             order[--i]=ptr->label;
             ptr=ptr->next;
```

```
        }
    }
    for(i=0;i<NNETS;i++) inverseorder[order[i]]=i;
    for(i=0;i<NNETS;i++) printf("%d ",order[i]);
    printf("\n");
    for(i=0;i<NNETS;i++) printf("%d ",inverseorder[i]);
    printf("\n");
    for(cptr=CA_START;cptr!=NULL;cptr=cptr->nextb)
    {
       cell=(cptr->name+1);
       ptr=cptr->nets;
       while (ptr!=NULL)
          {
             net1=ptr->label;
             ptr2=ptr->next;
             while (ptr2!=NULL)
             {
                   net2=ptr2->label;
                   smat[inverseorder[net1]][inverseorder[net2]]=cell;
                   smat[inverseorder[net2]][inverseorder[net1]]=cell;
                   ptr2=ptr2->next;
             }
              ptr=ptr->next;
        }
    }
    fprintmat(smat);
}
/*--------------------------------------*/
main()
{
   int **dmat;
   int *ordvec;
   int dmin,bsize,bmax,bcount,nmax,pass=0;
   Block *bestgroup,*bptr;

   /*construct cell_list and net_list from spice input file */
   input_data();
   //printout(NCELLS,NNETS);


   /* initialize dmat matrix */
   dmat = AllocInt2(NNETS,NNETS);
   makesc(dmat);
   //printblock(SC_START,"");

   do
```

```
{
   printf("Enter dmin :");
   scanf("%d",&dmin);
   ordvec=getord(dmin);
   maketc(dmin,ordvec);
   makesc(dmat);
   /*tc made from sc so need to re-do sc*/
   bsize=b_point_to(BORDER,GP_START)->size;
   if (bsize<=1) printf("dmin too large - try again.\n");
   if (bsize==NNETS) printf("dmin too small - try again.\n");
} while ((bsize<=1)||(bsize==NNETS));


//printblock(GP_START,"Group");
//printblock(TC_START,"");

printf("Enter nmax for initial grouping :");
scanf("%d",&nmax);
makegroups(nmax);
printf("Enter maximum block size required (>0) :");
scanf("%d",&bmax);

//printblock(TC_START,"TC");
//printblock(GP_START,"Group");
//printblock(CUT_START,"Cut ");

buildvertex();

//printblock(B_START,"Block ");
//printvertex(V_START,"Vertex ");

do
{
   printf("Pass %d\n",pass++);
   bestgroup=copy(GP_START);
   borderbalance();
   bsize=b_point_to(BORDER,GP_START)->size;
   nmax=0;
   bcount=0;
   for(bptr=B_START;bptr!=NULL;bptr=bptr->nextb)
   {
         bcount++;
         if (bptr->size>nmax) nmax=bptr->size;
   }
      //printblock(B_START,"Block ");
      //printvertex(V_START,"Vertex ");
```

```
    } while (nmax<=bmax);
    printblock(bestgroup,"Group");

    writematrix(bestgroup);
  }
```

A.3 Random graph generator

```
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#define RAND_MAX (pow(2,31)-1)
#define TRUE 1
#define FALSE 0
#define CELLMAX 1000

typedef struct list {
int node_id;
struct list *next;
}
LIST;

LIST *CA_start[CELLMAX], *CA_point[CELLMAX];

int D,N=500;

 /*----------------------------------------*/
LIST* insert(int thing, LIST *old_pointer)
  {
    LIST *pointer;

    pointer = (LIST *)malloc(sizeof(LIST));
    if (pointer == NULL)
    {
        printf("Not enough memory");
        exit(1);
    }
    pointer->node_id = thing;
    pointer->next = old_pointer;
    return(pointer);
}

 /*----------------------------------------*/
void printlist(int total)
{
  int i=0;
```

```
    FILE *out;

    out=fopen("mc.list","w");
    fprintf(out,"%d %d\n",total+1,N);
    for (i=0;i<=total;i++)
    {
      fprintf(out,"C%d ", i);

      for (CA_point[i] = CA_start[i]; CA_point[i] != NULL;
           CA_point[i] = CA_point[i]->next)
        fprintf(out,"%5d",   CA_point[i]->node_id);

    fprintf(out,"\n");
    }
    fclose(out);

    out=fopen("blockA.in","w");
    for (i=0;i<total;i=i+3)
    {
      fprintf(out,"C%d", i);
      if ((i+3) >= total) fprintf(out,"\n");
      else fprintf(out," ");
      /*if (i % 20 == 0) fprintf(out,"\n");*/
    }
    fclose(out);
}

 /*-----------------------------------------*/
int edge(double pr)
{
  int make_edge=FALSE;
  double temp_ran, nrand;

  temp_ran=random();
  nrand=temp_ran;
  nrand=(nrand/RAND_MAX);
  if (nrand<pr) make_edge=TRUE;

  return(make_edge);
}

 /*-----------------------------------------*/

main()
{
  int i,j,seed;
```

```
int edgecount=-1, in_list;
double pr;
double temp_ran, nrand;

printf("\nInput seed : ");
scanf("%d",&seed);
srandom(seed);
/*printf("\nInput # nodes in graph : ");
scanf("%d",&N);*/
printf("Input # edges required at each node (degree < %d) : ", N);
scanf("%d",&D);

pr = (double)D / ((double)N-1);

printf("\nProbability of making an edge  = %lf\n", pr);


for (i=0;i<N;i++)
{
  for (j=(i+1);j<N;j++)
  {
    if (edge(pr)==TRUE)
    {
      edgecount++;
      CA_start[edgecount]=insert(i,CA_start[edgecount]);
      CA_start[edgecount]=insert(j,CA_start[edgecount]);
    }
  }
}
/*check that all nodes are in the list. If not then add them*/
for (i=0;i<N;i++)
{
  in_list=0;
  for (j=0;j<edgecount;j++)
  {
    for (CA_point[j] = CA_start[j];
          CA_point[j] != NULL;
          CA_point[j] = CA_point[j]->next)
        if (CA_point[j]->node_id==i) in_list=1;
  }
  if (in_list==0)
  {
     do{
       temp_ran=random();
       nrand=temp_ran;
       nrand=(nrand/RAND_MAX)*N;
```

```
            j=(int)floor(nrand);
          } while (j==i);
          edgecount++;
          CA_start[edgecount]=insert(i,CA_start[edgecount]);
          CA_start[edgecount]=insert(j,CA_start[edgecount]);
        }
      }

      printlist(edgecount);
}
```

Appendix B

S functions

B.1 Geometric graph generator

```
mkgraph_function(n,d)
{
   coords <- cbind(runif(n), runif(n))
   distvec <- dist(coords)[1:sum(1:(n - 1))]
   nedges <- as.integer(d*n/2)
   evec_sort(order(distvec)[1:nedges])
   edgelist <- NULL
   x_1
   weight_0
   for(i in 1:nedges) {
      nxy_evec[i]
      while (nxy > (n-1)) {
         if (nxy < (n+weight)) nxy_nxy-weight
         else {
            x_x+1
            weight_weight+(n-x)
         }
      }
      edgelist <- rbind(edgelist, c(x,(nxy+1)))
   }
   tmp_sort(c(edgelist[,1],edgelist[,2]))
   nnodes_tmp[1]
   for(i in 2:len(tmp) ) {
   if(tmp[i]!=tmp[i-1]) nnodes_c(nnodes,tmp[i]) }
   singles_NULL
   if (nnodes[1]!=1) {
      nnodes_c(NA,nnodes)
      singles_c(singles,1)
   }
   for(i in 2:n ) {
      if (nnodes[i]=="NA") {
```

```
            nnodes_c(nnodes[1:(i-1)],NA)
            singles_c(singles,i)
         } else if (nnodes[i]!=i) {
            nnodes_c(nnodes[1:(i-1)],NA,nnodes[i:len(nnodes)])
            singles_c(singles,i)
         }
      }
      for(i in 1:len(singles)) {
         cat(singles[i],",")
         tdist_NULL
         if (singles[i]>1) {
            tdist_c(tdist,distvec[singles[i]-1])
         }
         if (singles[i]>2) {
            for (j in 2:(singles[i]-1)) {
               tdist_c(tdist,distvec[sum((n-1):(n-(j-1)))+(singles[i]-j)])
            }
         }
         if (singles[i]!=1) index_sum((n-1):(n-(singles[i]-1)))
         else index_0
         marker_len(tdist)
         tdist_c(tdist,distvec[(index+1):(index+(n-singles[i]))])
         j_order(tdist)[1]
         if (j>=marker) j_j+1
         edgelist <- rbind(edgelist, c(singles[i], j))
      }
      nedge_len(edgelist[,1])
      sink("mc.list")
      cat(nedge," ",n,"\n",sep="")
      for(i in 1:nedge) {
         cat("C",i," ",(edgelist[i,1]-1)," ",(edgelist[i,2]-1),"\n",sep="")
      }
      sink()
      return(list(edges=edgelist,coords=coords))
}

setup_function(nedge)
{
      sink("blockA.in")
      i_1
      while(i < nedge) {
         if (i==1) cat("C",i,sep="")
         else cat(" C",i,sep="")
         i_i+3
      }
      cat("\n")
```

```
    sink()
}
```

Appendix C

Mathematical models

C.1 Regression model

LINEAR MODEL BUILDING RESULTS

The Response Variable is 1

N= 50 NX= 12
SIGMAZ= 9.5611e-04 R2= 0.9997

NUMBER OF LINEAR MODEL PARAMETERS IS: 27

Variable	Beta	Std. Err.	t-val	R2-i
Constant	4.0965e+00	6.7102e-03	610.48	0.0000
/R215	2.7970e+00	2.1035e-02	132.97	0.5952
/R209	-1.7277e+00	2.2370e-02	-77.23	0.7239
/R213	-1.4267e+00	2.1940e-02	-65.03	0.7945
/R214	-1.3964e+00	2.0362e-02	-68.58	0.8693
/R210	8.9753e-01	2.0993e-02	42.75	0.9213
/R211	7.8512e-01	2.0970e-02	37.44	0.9695
/R215*/R214	-9.3690e-01	8.9731e-02	-10.44	0.9735
/R215*/R210	7.4863e-01	8.9638e-02	8.35	0.9777
/R209*/R209	-1.2634e-01	8.9498e-03	-14.12	0.9831
/R209*/R213	8.1913e-01	7.8751e-02	10.40	0.9870
/R215*/R209	-1.2868e+00	8.4072e-02	-15.31	0.9894
/R213*/R214	1.3919e+00	9.7335e-02	14.30	0.9917
/R215*/R213	-1.1625e+00	9.0873e-02	-12.79	0.9935
/R213*/R213	-4.5126e-02	6.9696e-03	-6.47	0.9947
/R215*/R211	3.7826e-01	9.9524e-02	3.80	0.9959
/R209*/R210	-5.3060e-01	7.5259e-02	-7.05	0.9975
/R214*/R214	-4.1297e-02	6.7094e-03	-6.16	0.9977
/R213*/C204	-2.7096e-01	8.4914e-02	-3.19	0.9979
/R209*/R211	-9.4715e-01	1.1232e-01	-8.43	0.9981
/R209*/R214	1.1489e+00	1.1366e-01	10.11	0.9988

/R209*/C204	-3.2996e-01	8.2138e-02	-4.02	0.9991
/R213*/R211	-4.6253e-01	9.7492e-02	-4.74	0.9991
/R213*/R210	-4.1318e-01	7.4361e-02	-5.56	0.9992
/R214*/R211	-6.5285e-01	1.2695e-01	-5.14	0.9994
/R214*/R210	-3.9022e-01	8.0440e-02	-4.85	0.9996
/R210*/R211	1.7634e-01	7.6862e-02	2.29	0.9997

Bibliography

[1] H L Abdel-Malek. The ellipsoidal technique for design centering and region approximation. *IEEE Trans. Computer Aided Design.*, 10:1006–1013, Aug 1991.

[2] H L Abdel-Malek and J W Bandler. Yield optimizations for arbitrary statistical distributions: Parts i & ii. *IEEE Trans. Circuits Syst.*, CAS-27:245–262, Apr 1980.

[3] P R Adby. *Applied Circuit Theory.* Ellis Horwood Limited, Chichester, 1980.

[4] D Agnew. Design centering and tolerancing via margin sensitivity minimization. *IEE Proc - G*, 127:270–277, Dec 1980.

[5] D Agnew. Improved minimax optimization for circuit design. *IEEE Trans. Circuits & Systs.*, CAS-28:791–803, Aug 1981.

[6] S B Akers. Clustering techniques for VLSI. *IEEE Trans. Computers*, 33(5):472–476, 1982.

[7] Antonio R Alvarez, Behrooz L Abdi, Dennis L Young, Harrison D Weed, Jim Teplik, and Eric R Herald. Application of statistical design and response surface methods to computer-aided VLSI device design. *IEEE Trans. Computer Aided Design*, 7:272–288, Feb 1988.

[8] K J Antreich, P Leibner, and F Pörnbacher. Nominal design of integrated circuits on circuit level by an interactive improvement method. *IEEE Trans. Circuits & Systs.*, 35:1501–1511, Dec 1988.

[9] P Balaban and J J Golembeski. Statistical analysis for practical circuit design. *IEEE Trans. Circuits & Systs.*, CAS-22:101–109, Feb 1975.

[10] J W Bandler and H L Abdel-Malek. Optimal centering tolerancing and yield determination via updated approximations and cuts. *IEEE Trans. Circuits & Systs.*, CAS-25:853–871, Oct 1978.

[11] J W Bandler, R M Biernacki, Qian Cai, S H Chen, Shen Ye, and Qi-Jun Zhang. Integrated physics-oriented statistical modelling, simulation and optimization. *IEEE Trans. Microwave Theory and Techniques*, 40(7):1374–1399, 1992.

[12] J W Bandler, S H Chen, and S Daijavad. Microwave device modelling using efficient l1 optimization: A novel approach. *IEEE Trans. Microwave Theory Tech.*, MTT-34:1282–1293, Dec 1986.

[13] J W Bandler, S H Chen, S Daijavad, and K Madsen. Efficient gradient approximations for non-linear optimization of circuits and systems. *Proc. IEEE Int. Symp. Circuits Syst.*, pages 964–966, 1986.

[14] J W Bandler, S H Chen, S Daijavad, and K Madsen. Efficient optimization with integrated gradient approximations. *IEEE Trans. Microwave Theory Tech.*, MTT-36:444–455, Feb 1988.

[15] J W Bandler and P C Liu. Automated network design with optimal tolerances. *IEEE Trans. Circuits & Systs.*, CAS-21:219–222, March 1974.

[16] J W Bandler, P C Liu, and J H K Chen. Worst case network tolerance optimization. *IEEE Trans. Microwave Theory & Tech.*, MTT-20:630–641, Aug 1975.

[17] J W Bandler, P C Liu, and H Tromp. A nonlinear programming approach to optimal design centering tolerancing and tuning. *IEEE Trans. Circuits & Systs.*, CAS-23:155–165, March 1976.

[18] John W Bandler and Shao Hua Chen. Circuit optimization : The state of the art. *IEEE Trans. Microwave Theory & Tech.*, 36:424–442, Feb 1988.

[19] John W Bandler and Qi-Jun Zhang. An automatic decomposition approach to optimization of large microwave systems. *IEEE Trans. Microwave Theory & Tech.*, 35:1231–1239, Dec 1987.

[20] John W Bandler, Qi-Jun Zhang, and Randoslaw M Biernacki. A unified theory for frequency-domain simulation and sensitivity analysis of linear and nonlinear circuits. *IEEE Trans. Microwave Theory & Tech.*, 36:1661–1454, Dec 1988.

[21] T B Barker. Quality engineering by design: Taguchi's philosophy. *Quality Assurance*, 13:72–80, Sept 1987.

[22] L Basso, A Winterbottom, and H P Wynn. A review of the 'taguchi methods' for off-line quality control. Technical Report 1, City University Statistical Laboratory, 1991.

[23] R A Bates, R J Buck, E Riccomagno, and H P Wynn. Experimental design and observation for large systems. *Journal of the Royal Statistical Society B*, to appear.

[24] R A Bates, R J Buck, and H P Wynn. Generic circuit response interpolation for robust design. *IEE Proc Circ. Dev. and Syst.*, to appear.

[25] R A Bates and H P Wynn. Tolerancing and optimization for model-based Robust Engineering Design. *Quality And Reliability Engineering International*, Submitted.

[26] Ron A Bates, R J Buck, and H P Wynn. Robust circuit design: An example. Technical Report 29, City University Engineering Design And Quality Centre, Northampton Square, London, 1991.

[27] Richard A Becker, J M Chambers, and A R Wilks. *The New S Language.* AT&T Bell Labs. Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, California, 1988.

[28] Maria C. Bernardo, Robert Buck, Lishin Liu, William A Nazaret, Jerome Sacks, and William J Welch. Integrated circuit design optimization using a sequential strategy. *IEEE Trans. Computer Aided Design.*, CAD-11:361–372, 1992.

[29] G E P Box and D W Behnken. Some new three level designs for the study of quantitative variables. *Technometrics*, 2:455–475, 1960.

[30] G. E. P. Box and S Jones. Designing products that are robust to the environment. *Total Quality Management*, in press, 1992.

[31] George Box, Soren Bisgaard, and Conrad Fung. An explanation and critique of taguchi's contributions to quality engineering. *Quality and Reliability Engineering International*, 4:123–131, 1988.

[32] Franklin H Branin. Computer methods of network analysis. *Proceedings of the IEEE*, 55:1787–1801, Nov 1967.

[33] R K Brayton and R Spence. *Sensitivity and optimization.* Elsevier, Amsterdam, 1980.

[34] R K Brayton and R Spence. *Sensitivity and optimization*, chapter 6, page 134. Elsevier, Amsterdam, 1980.

[35] R J Buck. Robust Engineering Design Users Guide. Technical report, City University Engineering Design And Quality Centre, Northampton Square, London, 1993.

[36] R J Buck. *The design and analysis of computer experiments.* PhD thesis, City University, London, UK, 1994.

[37] R J Buck and H P Wynn. Optimization strategies in robust engineering design and computer-aided design. *Quality And Reliability Engineering International*, 9:39–48, 1993.

[38] R J Buck and H P Wynn. Improving the distribution of points in a Latin Hypercube sample. *Technometrics*, submitted.

[39] E M Butler. Large change sensitivities for statistical design. *Bell Syst. Tech. J*, 50:1209–1224, April 1971.

[40] E M Butler. Realistic design using large change sensitivities and performance contours. *IEEE Trans. Circuit Theory*, CT-18:58–66, Jan 1971.

[41] Frank M Callier, Wan S Chan, and Charles S Desoer. Input-output stability theory of interconnected syatems using decomposition techniques. *IEEE Trans. Circuits & Systems*, CAS-23(12):714–729, Dec 1976.

[42] C Charalambous and M El-Turky. Circuit design using a recent minimax approach. *Computer Aided Design*, 11(1):27–31, Jan 1979.

[43] L K Chen, B S Ting, and A Sangiovanni-Vincentelli. An edge-oriented adjacency list for undirected graphs. *Int. J Circ. Theory & Appl.*, 7:55–63, 1979.

[44] Chung-Kuan Cheng. The optimal partitioning of networks. *NETWORKS*, 22:297–315, 1992.

[45] Leon O. Chua and Pen-Min Lin. *Computer Aided Analysis of Electronic Circuits : algorithms and computational techniques.* Prentice Hall Inc., Englewood Cliffs, New Jersey, 1975.

[46] R M Cormack. A review of classification. *Journal Royal Stat. Soc. B*, pages 321–353, Mar 1971.

[47] Noel Cressie. *Statistics for Spatial Data*, chapter 2, page 101. John Wiley and Sons, Inc., 1991.

[48] An-Cheng Deng. On network partitioning algorithm of large-scale cmos circuits. *IEEE Trans. Circuits & Systs.*, 36:294–299, Feb 1989.

[49] Madhav P Desai and Ibrahim N Hajj. On the convergence of block relaxation methods for circuit simulation. *IEEE Trans. Circuits & Systs.*, 36:948–958, July 1989.

[50] C Desoer and E Kuh. *Basic Circuit Theory*, page 654. McGraw-Hill, New York, 1969.

[51] S W Director and G D Hachtel. The simplicial approximation approach to design centering. *IEEE Trans. Circuits & Systs.*, CAS-24:363–372, July 1977.

[52] S W Director and R A Rohrer. The generalized adjoint network and network sensitivities. *IEEE Trans. Circuit Theory*, CT-16:318–323, Aug 1969.

[53] Stephen W Director. Manufacturing-based simulation : An overview. *IEEE Circuits and Devices Magazine*, pages 3–9, Sep 1987.

[54] T Eckstein and E Luder. Design centering by improved monte carlo analysis of the region of acceptability. *Proc. IEEE Int. Symposium Circ. Syst.*, pages 951–954, 1986.

[55] H Eltahawy, S Garciasabiro, D Rodriguez, and J J Mayol. Towards an analog hardware description language - based on VHDL. In *Proceedings of the 1994 Western Multiconference*, pages 48–53. International Conference on Simulation and Hardware Description Languages (ICSHDL), 1994.

[56] Walter L Engl, Rainer Laur, and Heinz K Dirks. MEDUSA - a simulator for modular circuits. *IEEE Trans. Computer Aided Design.*, CAD-1:85–93, Apr 1982.

[57] Brian Everitt. *Cluster Analysis*, volume Second Edition. Halsted Press, New York, 1981.

[58] N. Fenton and G. Hill. *Systems, construction and analysis.* McGraw-Hill, London, 1993.

[59] C M Fiduccia and R M Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181, 1982.

[60] M R Garey and D S Johnson. *Computers and Interactability : a guide to the theory of NP-completeness.* W H Freeman and Co., 1979.

[61] Georges E Gielen and Herman C C Walscharts. Isacc : A symbolic simulator for analog integrated circuits. *IEEE Journal of Solid State Circuits*, 24:1587–1597, Dec 1989.

[62] Georges E Gielen, Herman C C Walscharts, and Willy C Sansen. Analog circuit design optimization based on symbolic simulation and simulated annealing. *IEEE Journal of Solid State Circuits*, 25:707–713, June 1990.

[63] Gary D Hachtel and Alberto L Sangiovanni-Vincentelli. A survey of third-generation simulation techniques. *Proceedings of the IEEE*, 69:1264–1280, Oct 1981.

[64] Ibrahim N Hajj. Sparsity considerations in network solution by tearing. *IEEE Trans. Circuits & Systs.*, CAS-27:357–366, May 1980.

[65] D S Harrison, A R Newton, D L Spickelmier, and T J Barnes. Electronic CAD frameworks. *IEEE Proceedings.*, 78(2):393–417, 1990.

[66] J A Hartigan. Distribution problems in clustering. In J Van Ryzin, editor, *Classification and Clustering*, pages 45–71. Academic Press, Inc., London, 1977.

[67] M R Irving and M J H Stirling. Optimal network tearing using simulated annealing. *IEE Proceedings Pt-C*, 137:69–72, Jan 1990.

[68] L Jaulin and E Walter. Guaranteed nonlinear parameter-estimation from bounded-error data via interval-analysis. *Mathematics and Computers in Simulation*, 35(2):123–137, 1993.

[69] L Jaulin and E Walter. Set inversion via interval-analysis for nonlinear bounded-error estimation. *Automatica*, 29(4):1053–1064, 1993.

[70] A. Jebb and H P Wynn. Robust engineering design post-taguchi. *Phil. Trans. R. Soc. London A*, 327:605–616, 1989.

[71] F Branin Jr. Network sensitivity and noise analysis simplified. *IEEE Trans. Circuit Theory*, CT-20:285–288, May 1973.

[72] R N Kackar. Off-line quality control parameter design and the taguchi method. *Journal Of Quality Technology*, 17:176–188, Oct 1985.

[73] B J Karafin. The optimum assignment of component tolerances for electrical networks. *Bell Syst. Tech. J*, 50:1225–1243, Apr 1971.

[74] S Kirkpatrick, C D Gelatt, and M P Vecchi. Optimization by simulated annealing. *Science*, 220:671–679, May 1983.

[75] Balakrishnan Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. Computers*, C-33:438–446, May 1984.

[76] G Kron. *Diakoptics: The Piecewise Solution of Large Scale Systems.* MacDonald, London, 1963.

[77] G Kron. *Diakoptics: The Piecewise Solution of Large Scale Systems*, page 6. MacDonald, London, 1963.

[78] Kenneth S Kundert and Alberto Sangiovanni-Vincentelli. Simulation of nonlinear circuits in the frequency domain. *IEEE Trans. Computer Aided Design*, CAD-5:521–535, Oct 1986.

[79] A Liberatore and S Manetti. SAPEC - a personal computer program for the symbolic analysis of electric circuits. *Proceedings IEEE Symposium On Circuits & Systs.*, pages 897–900, 1988.

[80] Dennis K J Lin and Norman R Draper. Projection properties of plackett and burman designs. *Technometrics*, 34:423–428, 1992.

[81] P M Lin. A survey of applications of symbolic network functions. *IEEE Trans. Circuit Theory*, CT-20:732–737, Nov 1973.

[82] K K Low and Stephen W Director. An efficient methodology for building macromodels of ic fabrication processes. *IEEE Trans. Computer Aided Design.*, CAD-8:1299–1313, Dec 1989.

[83] K K Low and Stephen W Director. A new methodology for the design centering of ic fabrication processes. *IEEE Trans. Computer Aided Design*, 10:895–903, July 1991.

[84] F Luccio and M Sami. On the decomposition of networks in minimally interconnected subnetworks. *IEEE Trans. Circuit Theory*, 16(2):184–188, 1969.

[85] M. D. McKay, W J Conover, and R J Beckman. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21:239–245, 1979.

[86] G J Meidt and K W Bauer Jr. Pcrsm :'A decision support system for simulation metamodel construction. *Simulation*, 59(3):183–191, 1992.

[87] M. D. Mesarovic, D. Macko, and Y. Takahara. *Theory of Hierarchical, Multilevel Systems.* Academic Press, New York, 1970.

[88] Glenn W Milligan. A monte carlo study of thirty internal criterion measures for cluster analysis. *Psychometrica*, 46:187–199, June June 1981.

[89] T. Mitchell, J. Sacks, and D. Ylvisaker. Asymptotic bayes criteria for nonparametric response surface design. *Ann. Statist.*, 22, No. 2, 1994 (to appear).

[90] M. D. Morris, T. J. Mitchell, and D. Ylvisaker. Bayesian design and analysis of computer experiments: use of derivatives in surface prediction. *Technometrics*, 35(3):243–255, Aug 1993.

[91] Peter Mucci. *Handbook for engineering design using standard materials and components.* P E R Mucci Ltd., The Old Bakery, Parsonage Lane, Durley, Southampton SO3 2AD, 1986.

[92] Tamal Mukherjee and L Richard Carley. Rapid yield estimation as a computer aid for analog circuit design. *IEEE Journal of Solid State Circuits*, 26:291–299, Mar 1991.

[93] R. H. Myers. Response surface methodology in quality improvement. *Communications in Statistics-Theory and Methods*, 20(2):457–476, 1991.

[94] L W Nagel. *SPICE 2. A computer program to simulate semiconductor circuits.* ERL Memo ERL-M520. Univ. California, Berkley, 1975.

[95] S R Nassif, A J Strojwas, and S W Director. FABRICS II : a statistically based IC fabrication process simulator. *IEEE Trans. Computer-Aided Design*, 3, 1984.

[96] Arthur Richard Newton and Alberto L Sangiovanni-Vincentelli. Relaxation based electrical simulation. *IEEE Trans. Electron Devices*, ED-30:1184–1207, Sep 1983.

[97] K G Nichols, T J Kazmierski, M Zwolinski, and A D Brown. Overview of SPICE-like circuit simulation algorithms. *IEE Proc.-Circuits Devices Syst.*, 141(4):242–250, 1994.

[98] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* CBMS-NFS, SIAM, Philadelphia, 1992.

[99] E C Ogbuobiri, W F Tinney, and J W Walker. Sparsity-directed decomposition for gaussian elimination on matrices. *IEEE Trans. Power Apparatus & Systems.*, PAS-89:141–150, Jan 1970.

[100] J H O'Geran. *Group testing and search.* PhD thesis, City University, London, UK, 1994.

[101] A. B. Owen. A central limit for latin hypercube sampling. *Jour. Roy. Statist. Soc., Serie B*, 1992.

[102] A. B. Owen. Lattice sampling revisited: Monte Carlo variance of means over randomized orthogonal arrays. *Ann. Statist.*, 22, 1994 (in press).

[103] Donald O Pederson. A historical review of circuit simulation. *IEEE Trans. Circuits & Systs.*, CAS-31:103–111, Jan 1984.

[104] P Penfield Jr, R Spence, and S Duinker. A generalized form of tellegen's theorem. *IEEE Trans. Circuit Theory*, 4:302–305, 1953.

[105] R. L. Plackett and J. P. Burman. The design of optimum multifactorial experiments. *Biometrika*, 33:305–325, 1946.

[106] J. Sacks, S. B. Schiller, and W. J. Welch. Designs for computer experiments. *Technometrics*, 31:41–47, 1989.

[107] Jerome Sacks, William J Welch, Toby J Mitchell, and Henry P Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–435, Nov 1989.

[108] Alberto Sangiovanni-Vincentelli, Li-Kua Chen, and Leon O Chua. An efficient heuristic cluster algorithm for tearing large-scale networks. *IEEE Trans. Circuits & Systs.*, CAS-24:709–717, Dec 1977.

[109] A Sarmiento Reyes. Efficient pertitioning-based method to determine the upper bound on the number of operating points in transistor circuits. *IEE Trans. Circuits and Systems*, 141(4):258–264, 1994.

[110] J D Schoeffler. The synthesis of minimum sensitivity networks. *IEEE Trans. Circuit Theory*, CT-11:271–276, June 1964.

[111] Anne C. Shoemaker, Kwok-Leung Tsui, and C. F. Jeff Wu. Economical experimentation methods for robust design. *Technometrics*, 33:415–428, 1991.

[112] Mark R Simpson. Pride : An integrated design environment for semiconductor device simulation. *IEEE Trans. Computer Aided Design*, 10:1163–1174, Sep 1991.

[113] K Singhal and J F Pinel. Statistical design centering and tolerancing using parametric sampling. *IEEE Trans. Circuits Syst.*, CAS-28:692–701, July 1981.

[114] Kishore Singhal, Colin C McAndrew, Sani R Nassif, and V Visvanathan. The CENTER design optimization system. *AT&T Technical Journal*, pages 77–91, May 1989.

[115] Kishore Singhal and Jiri Vlach. Symbolic analysis of analog and digital circuits. *IEEE Trans. Circuits & Systs.*, CAS-24:598–609, Nov 1977.

[116] R S Soin and R Spence. Statistical exploration approach to design centering. *IEEE Proc-G*, 127:260–269, Dec 1980.

[117] James P Spoto, W Terry Coston, and C Paul Hernandez. Statistical integrated circuit design and characterization. *IEEE Trans. Computer Aided Design*, CAD-5:90–103, Jan 1986.

[118] J Starzyk. Signal-flow-graph analysis by decomposition method. *IEE Proceedings Pt-G*, 127:81–86, Apr 1980.

[119] J A Starzyk and E Sliwa. Tolerances in symbolic network analysis. *Proc. IEEE Int. Symp. Circuits & Systems*, 2:810–3, 1989.

[120] Janusz Starzyk and Edward Sliwa. Hierarchic decomposition method for the topological analysis of electronic networks. *Circuit Theory And Applications*, 8:407–417, 1980.

[121] Janusz A Starzyk and A Konczykowska. Flowgraph analysis of large electronic networks. *IEEE Trans. Circuits & Systs.*, CAS-33:302–315, Mar 1986.

[122] M. Stein. Large sample properties of simulations using latin hybercube sampling. *Technometrics*, 29:143–151, 1987.

[123] M L Stein. An efficient method of sampling for statistical circuit design. *IEEE Trans. Computer Aided Design*, CAD-5:23–29, Jan 1986.

[124] M A Styblinski. Problems of yield gradient estimation for truncated probability density functions. *IEEE Trans. Computer Aided Design.*, CAD-5:30–38, Jan 1986.

[125] Fang K T and Wang Y. *Number-theoretic Methods in Statistics.* Chapman & Hall, London, 1994.

[126] G Taguchi. Off-line and on-line quality control systems. In *ICQC*, pages B4.1–B4.5., Tokyo, 1978.

[127] G Taguchi. *System Of Experimental Design: engineering methods to optimize quality and minimize costs*, volume 1. Unipub / Kraus International Publications, White Plains, New York, 1987.

[128] G Taguchi and M S Phadke. Quality engineering through design optimization. In *IEEE Globe 1984 Conference Atlanta GA*, volume 3, pages 1106–1113, Nov 1984.

[129] K S Tahin and R Spence. A radial exploration approach to manufacturing yield estimation and design centering. *IEEE Trans. Circuits & Systs.*, CAS-26:768–774, Sept 1979.

[130] L Tao and Y C Zhao. Effective heuristic algorithms for VLSI circuit partition. *IEE Proceedings-G*, 140(2):127–134, April 1993.

[131] P Tatjewski, N Abdullah, and P D Roberts. Comparison of some algorithms for hierarchical steady-state optimizing control of interconnected industrial-processes. In *International Conference on Control 88*, volume 285, pages 527–531, 1988.

[132] B D H Tellegen. A general network theorem with applications. *Proc. Inst. Radio Engineers, Australia*, 14:265–270, 1953.

[133] Gabor C Temes and Donald A Calahan. Computer-aided network optimization the state of the art. *IEEE Proceedings*, 55:1832–1863, Nov 1967.

[134] J. F. Traub, G. W. Wasilkowski, and H. Woźniakowki. *Information-Based Complexity.* Academic Press, New York, 1988.

[135] M E Van Valkenburg. *Analog Filter Design.* CBS College Publishing, New York, 1982.

[136] G Geoffrey Vining and Raymond H Myers. Combining taguchi and response surface philosophies : A dual response approach. *Journal of Quality Technology*, 22:38–44, Jan 1990.

[137] Paul K U Wang, Chin Fu Chen, and Yuang-Sheng Kao. Sensitivity calculation and network optimization through decomposition. *Proc. IEEE Int. Symp. Circuits & Systems*, 3:1034–7, 1983.

[138] G W Wasilkowski and H Woźniakowski. There exists a linear problem with infinite combinatory complexity. *Journal of Complexity*, 9:326–337, 1993.

[139] W. J. Welch, R. J. Buck, J. Sacks, H. P. Wynn, T. J. Mitchell, and M. D. Morris. Screening, predicting, and computer experiments. *Technometrics*, 34(1):15–25, 1992.

[140] W. J. Welch and J. Sacks. A system for quality improvement via computer experiments. *Communications in Statistics-Theory and Methods*, 20(2):477–496, 1991.

[141] W. J. Welch, Tat-Kuan Yu, S M Kang, and J Sacks. Computer experiments for quality control by parameter design. *Journal of Quality Technology*, 22:15–22, 1990.

[142] Jőrg Wintermantel. Optimization of real-coefficient solutions of multiple-feedback digital filters. *Proccedings of IEEE ISCAS*, pages 947–948, 1986.

[143] H P Wynn, R A Bates, R J Buck, and A Carter. A statistical bolt-on for robust circut design. *Journal of Design and Manufacturing*, 4:81–86, 1994.

[144] H. P. Wynn and A. A. Zhigljavsky. *Fundamentals of Search.* Springer-Verlag, New York, to appear.

[145] S Yeşilyurt, C K Ghaddar, M E Cruz, and A T Patera. Bayesian-validated surrogates for noisey computer simulations; application to random media. *SIAM journal on scientific computing*, To appear.

[146] S Yeşilyurt and A T Patera. Surrogates for numerical simulations; optimization of eddy-promoter heat exchangers. *Computer methods in applied mechanics and engineering*, 1994.

[147] Jih-Shyr Yih and Pinaki Mazumder. A neural network design for circuit partitioning. *IEEE Trans. Computer Aided Design.*, CAD-9:1265–1271, Dec 1990.

[148] Z. Ying. Maximum likelihood estimation of parameters under a spatial sampling scheme. *Ann. Statist.*, 21(3):1567–1590, 1993.

[149] D Ylvisaker. Prediction and design. *The Annals of Statistics*, 15:1–19, 1987.

[150] Dennis L Young, Jim Teplik, Harrison D Weed, Neil T Tracht, and Antonio R Alvarez. Application of statistical design and response surface methods to computer-aided VLSI device design ii : Desirability functions and taguchi methods. *IEEE Trans. Computer Aided Design*, 10:103–115, Jan 1991.

[151] T. K. Yu, S M Kang, J Sacks, and W J Welch. Parametric yield optimization of mos integrated circuits by statistical modeling of circuit performances. *International Journal of Circuit Theory and Applications*, in press, 1991.

[152] Peter J Zemroch. Cluster analysis as an experimental design generator, with ăpplication to gasoline blending experiments. *Technometrics*, 28:39–49, Feb 1986.

[153] A I Zečevič and D D Šiljak. Balanced decompositions of sparse systems for multilevel parallel processing. *IEEE Trans. Circuits & Systems-I: Fundamental Theory & Applications.*, 41(3):220–233, March 1994.

[154] J Zhang and J W Modestino. A model-fitting approach to cluster validation with application to stochastic model-based image segmentation. *IEEE Trans. Pattern Analysis & Machine Intelligence*, 12:1009–1017, Oct 1990.

[155] Anatoly A. Zhigljavsky. *Theory of Global Random Search*, chapter 4. Kluwer Academic Publishers, 1991.